A Constructive Neural Network Incorporating Competitive Learning of Locally Tuned Hidden Neurons

by

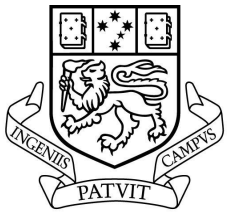
Simon William D'Alton, BComp

A dissertation submitted to the

School of Computing

in partial fulfilment of the requirements for the degree of

Bachelor of Computing with Honours



University of Tasmania

November, 2005

Declaration of Originality

This thesis contains no material which has been accepted for the award of any other degree or diploma in any tertiary institution, and that, to my knowledge and belief, this thesis contains no material previously published or written by another person except where due reference is made in the text of the thesis.

.....

Simon D'Alton

Abstract

Performance metrics are a driving force in many fields of work today. The field of constructive neural networks is no different. In this field, the popular measurement metrics (resultant network size, test set accuracy) are difficult to maximise, given their dependence on several varied factors, of which the most important is the dataset to be applied.

This project set out with the intention to minimise the number of hidden units installed into a resource allocating network (RAN) (Platt 1991), whilst increasing the accuracy by means of application of competitive learning techniques. Three datasets were used for evaluation of the hypothesis, one being a time-series set, and the other two being more general regression sets.

Many trials were conducted during the period of this work, in order to be able to prove conclusively the discovered results. Each trial was different in only one respect from another in an effort to maximise the comparability of the results found. Four metrics were recorded for each trial- network size (per training epoch, and final), test and training set accuracy (again, per training epoch and final), and overall trial runtime.

The results indicate that the application of competitive learning algorithms to the RAN results in a considerable reduction in network size (and therefore the associated reduction in processing time) across the vast majority of the trials run. Inspection of the accuracy related metrics indicated that using this method offered no real difference to that of the original implementation of the RAN.

As such, the positive network-size results found are only half of the bigger picture, meaning there is scope for future work to be done to increase the test set accuracy.

Acknowledgements

There are several people who I would like to acknowledge my appreciation for over the course of this research project.

Firstly I would like to thank my family; thanks Mum, Dad and Jeremy for letting me do my own thing at your expense, and for appreciating the pressure I have endured for the final weeks of this project.

Secondly I would like to thank my supervisor, Peter. Thanks for coming through with suggestions when things weren't going my way.

Thirdly, and most publicly, I would like to thank all the other honours students for just being around. Without you guys honours would not be as nearly as enjoyable as it has been, although my time may have been spent more productively.

Honourable mentions go to:

- The bloke next door, Duncan; for forever keeping me updated with latest news, and to the rest of the office for providing oftentimes inappropriate, but amusing commentary on the work at hand.
- Ivan for selflessly and extensively proofreading my work at the 11th hour.
- The honours students in my research group: thanks Stephen, Dimitry and Emma for letting me bounce ideas off you.
- My close mates for being around and letting me unwind.
- The school for providing me with the opportunity to do my degree, and also for giving me access to massive parallel computation which went a long way to calculate those all important final results.

Table of Contents

List of Figures

List of Tables

List of Equations

1 Introduction

Much work has been done in the field of neural networks since the innovation of the perceptron in 1958 (McCulloch and Pitts 1943; Rosenblatt 1958). However, no single approach to machine learning has been shown to be successful for all sets of data.

Unfortunately in the current day, there is no such thing as a perfect machine learning algorithm, nor is it certain that there ever will be. For a learner to be considered successful, attention must be paid to the characteristics of the data that will be used in the system. Neural networks have been applied successfully to the broad field of data mining before, and are known to be able to be applied to most types of data. Neural networks have two main problems:

a) They usually require a large amount of time to be trained on a particular dataset before the algorithm is to have any accuracy on its intended application; and

b) They have a tendency to over fit the data. That is, they become very accurate on the data that they have access to, at the expense of accuracy of data that has not yet been seen. This can be compared to humans rote learning something, as opposed to learning the underlying theory of the problem.

The overarching goal of this research project is to modify a particular algorithm to enable it to use competitive learning techniques. It is hypothesized that this addition may result in a decrease in the training time required, and form a more compact representation of the dataset. The Resource Allocating Network (RAN) is an implementation of the aforementioned algorithm, and forms the basis for this research project. In its current state, the RAN is adept at learning only time series datasets. It is hoped that this can be extended successfully with the

application of competitive learning to in order to cope with the more general function approximation datasets that are also considered in this research project.

2 Literature Review

2.1 Neural Networks

The Artificial Neural Network (ANN) is a biologically inspired approximated model of the human brain. The ANN model achieves its power by mimicking the behaviors of the brain's neurons. Artificial neurons consist of three general parts: inputs, processing and outputs, which when combined are representative of the general function of a biological neuron.

In a biological environment, neurons propagate one-way electrical signals throughout the brain. In order to propagate, each neuron is connected directly with the outputs of other neurons, and has other neurons connected to its output. This network architecture allows each neuron to aggregate the output of its input-connected neighbours. With the ability to generalise, the power of the brain is realized- each neuron can generalise any number of other input-neurons, which in turn can generalise any other neuron. The complex connection networks that result allow for a controlled propagation of any given stimulus to some specialised neuron which can then act by triggering a reaction, a thought, a memory etc.

2.1.1 The perceptron

The basic artificial neuron is called the perceptron and was designed to mimic the features of a biological neuron (Rosenblatt 1958). The neuron can accept any number of inputs so long as the values they carry are binary (That is, their values are either 0 or 1). The output is calculated by comparing the weighted sum of the inputs to a threshold value. Should the weighted sum exceed the threshold, the perceptron will 'fire' (output) a value of 1, otherwise a value of 0 will be fired. In 1969, Minsky and Papert published a paper demonstrating the

perceptrons inability to successfully solve the XOR problem, as the problem space could not be split in two (Minsky and Papert 1969).

2.1.2 Connection Weights

Despite their flexible learning capabilities, the only adjustable part of multi layer perceptron networks is the weights of the connections between neurons. Through the process of training, these weights are gently raised or lowered by the training process, in order to emphasize the importance of a particular connection. Extra connections are installed into many architectures, named the bias (or offset). This connection allows for further flexibility in learning, as it is able to learn an offset to a particular problem, in allowing the neuron a finer degree of precision in which to operate.

2.1.3 Transfer Functions

It was found that different functionality could be exhibited if the threshold functions of the perceptron were altered. Research has found that applying different mathematical functions to the neuron processor can give different responses, which if used in the right environment, can be quite effective. For instance, many neural networks have a sigmoidal transfer function in their input and hidden layers, and a linear transfer unit for the output layer. Some functions that are commonly used are described in Equation 2.1, Equation 2.2 and Equation 2.3 below.

$$P(t) = t$$

Equation 2.1: Linear transfer function

$$P(t) = \frac{1}{1 + e^{-t}}$$

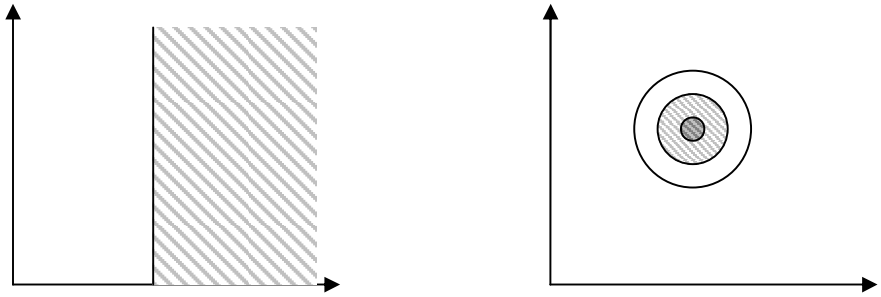
Equation 2.2: Sigmoid transfer function

$$P(t) = e^{-t^2}$$

Equation 2.3: Gaussian transfer function

Locally Tunable Response

Gaussian transfer functions are also known as a 'local' transfer function. This is because the function itself starts and ends at the same asymptote, allowing for a radial partitioning of space rather than an axis aligned boundary. This means there is no a injective nature of the transfer function. This contrasts with non-locally tunable transfer functions in where this mapping does exist. This is demonstrated in Figure 2.1.



(a) Sigmoidal Response (a) Gaussian Response

Figure 2.1: Effect of adding a neuron in a two attribute environment

Limitations of globalised response

Usage of only non-locally responsive transfer functions (that is, globally responsive) can limit the effective size of the resultant network. The one-to-one mapping of the transfer function is the reason behind this- it enables the previously connected units to summarise the input by using the connection weights to effectively implement a limit in the signal strength. However, this effect can be negated by using locally responsive neurons in the network in such a way that chains of more than two globally responsive neurons are broken. Another method, which is used in cascade correlation networks, is to cease training connection weights after a new unit is added.

2.1.4 Pitfalls of Neural Networks

Over training

> "Over-training a neural network is similar to when an athlete practices and practices for an event on his home court. When the actual competition starts and he or she is faced with an unfamiliar arena and circumstances it might be impossible for him or her to react and perform at the same levels as during training." (Bao 2000, p. 94)

Over training can occur when a training dataset is of limited size, or the network is presented with the dataset too many times, allowing the network to memorise the set on a point by point basis. This is a problem, as when the network has the testing set presented to it, the (usually) slightly different set will have a high degree of error due to the dataset 'response optimisation' that occurred during training. To prevent this problem from occurring, it is generally accepted that the learning be terminated after a defined amount of training epochs, an achieved error, or a maximum size that it can grow to.

Training Speed of Backpropagation

Backpropagation is a simple yet effective way to train neural networks (Rumelhart and Zipser 1985; Werbos 1974). The name backpropagation refers to how the error is computed for each unit in the network. When an input pattern is received, each receiving node calculates their response by doing a weighted sum of each of its inputs, and applying it transfer function to this. This response is pushed forwards through the network until the signal reaches the output layer. This process is the same for all feed forward neural networks. In backpropagation trained networks, the error is then computed at the output units and sent backwards through the network where each unit will calculate its own error term, and update its own weights. This process is slow because the algorithm can only move in fixed size steps down the error gradient computed.

The choice of step size also needs to be made, the smaller the value the more accurate the model can end up, but the easier it is to get caught in local minima. The larger the value, the quicker the network will find a solution, but is equally likely to miss the solution altogether (stepping right over the solution).

2.2 Network Topology

> "In order to achieve good generalization performance when training a neural network, it must have the right size. Networks that are too small cannot represent the required function, while networks that are too large are prone to over fitting… To limit the effective size of the network in order to avoid over fitting one can either use additive [constructivist] or subtractive [destructive] methods or regularization." (Prechelt 1996, p. 2)

2.2.1 Network Size Management

Neural networks can be said to subscribe to one of three paradigms with respect to size maintenance. Each of the three general paradigms are explained below.

Fixed Architecture Algorithms

Static networks can be described as not being able to dynamically change their network topology. Two examples of fixed size networks are multi-layer perceptrons and Radial Basis Function networks (RBF) (Moody and Darken 1989). Static sized networks have the potential to be more computationally efficient than dynamic size networks as they do not need to constantly check whether to add a new unit or remove an existing unit. Having a predefined network topology can also be detrimental to performance. In the first instance an expert needs some way to come up with an appropriate network size and structure, which is heavily dependent on the particular application of the network. Should the application of the network change, or the data change in format, syntax , frequency distribution, or type, the model may suffer a

decrease in accuracy or increase in time taken to learn- requiring the model to be again manually adjusted. Although this is true for any kind of machine learning algorithm, it is particularly prevalent in fixed dimensionality networks.

Destructive Algorithms

The aim of destructive algorithms is to remove underachieving nodes or connections from the network. Connections can be either nullified by having their weights set to 0, or removed entirely from the network. In order to ascertain what to remove from the network one must examine the entire network in order to see what components have the greatest unwanted influence on network performance. The use of saliency measures allows algorithms to measure the relevance of a network connection. This measurement can then be compared to other connection's saliency measures, allowing the least performing connection to be removed. An alternative is that all connections that have saliencies below zero (which are doing the network more harm than good) to be removed.

One example of a pruning algorithm is that of Optimal Brain Damage (OBD) (Cun et al. 1990) which is a method that estimates the change in error for a change in a set of weights. Other well known pruning methods are principal components pruning (PCP) (Levin et al. 1994) and optimal brain surgeon (OBS) (Hassibi and Stork 1993).

Should one wish to nullify connections instead (rendering them useless by setting the weight to zero), one can opt for methods such as weight decay. Weight decay (Weigend 1991) involves the gradual reduction in the weights magnitude when it is not being used. This mechanism allows the network to 'undo' its actions as connections are never explicitly removed, although it incurs a performance hit over explicitly purging connections. In contrast, removing connections altogether has the potential to decrease the amount of calculation

the network has to do, but the ability to reinstate connections must be handled in a manual way.

Destructive methods can be applied to any type of neural network, but are typically applied to constructive neural networks, where a node or connection can be created to replace the removed weight or relocate it after it has been disassociated with the network.

Constructive Algorithms

> "Algorithms such as Back Propagation can suffer from catastrophic interference when learning new data… storage of new information seriously disrupts the retrieval of previously stored data. The incremental learning strategy of constructive algorithms offers a possible solution at this time."(Campbell 1997, p. 3)

Constructive algorithms take their name from the way they incorporate new knowledge into the network. A constructive algorithm will grow the network structure against the training examples. Typically, constructive algorithms will start with a one layer perceptron style network, with input neurons being fully connected to output neurons. When being trained, neurons are inserted into the network. Two examples of well known constructive networks are Cascade Correlation (Fahlman and Lebiere 1990) and the Resource-Allocating Network (RAN) (Platt 1991).

It is possible for constructive algorithms to have destructive aspects applied to them; similarly, destructive methods are able to implement constructive techniques. Care must be taken when using both constructive and destructive techniques as networks can get caught in oscillatory behavior. This may be instigated by the creation of a new unit, leading the algorithm to adjust the network to make it fit, then observing that the new unit was useless, and so promptly removing that unit. The net effect for this sequence of actions is zero,

but it is processing lost which could have gone into more constructive changes to the network.

Constructive techniques have the ability to learn the complexity of the training data, and to install the minimum amount of units to suit. Because of their incremental nature, smaller solutions are tried before larger ones giving the result of a typically smaller network (Parekh et al. 2000). Constructive algorithms also allow tradeoffs to be made (Kwok and Yeung 1995) between different measures. For example, the tradeoff between network size and generalization accuracy.

2.3 Network Considerations

When selecting or training a neural network several decisions must be made about the application of the network. These decisions are heavily dependent on the input data, and its representation and also the desired application of the network.

2.3.1 Online versus Offline Learning

When it comes time to train a network, it needs to be decided if training will be performed either online or offline over the data set. Justifications for which one should be selected, is made with respect to how much data the network needs to have access to. Offline learning can be used when datasets are of a finite and manageable size, and when data patterns need to be presented to the network on a number of occasions. Vice versa, online learning is a process where each data pattern is only presented to the network once. The information flow into an online algorithm can be visualized as a stream, you only know about the present and the immediate past, whereas the information flow for an offline algorithm can be conceptualized as a book that can be read repeatedly. Online learning permits an alternative to the offline learning paradigm should the dataset be of substantially large size, or even unlimited size. That is, online

learning is used when it becomes impractical or impossible to iterate over the dataset multiple times, or for real time incremental learning.

2.3.2 Supervised versus Unsupervised Learning

Supervision in learning reflects the nature of the dataset that is in use. Datasets that include the expected resultant behavior, or solution for each input pattern can be trained by supervised learning methods. Conversely, unsupervised methods, can be used when output information is not included in the dataset. One can differentiate between supervised and unsupervised learning methods since supervised methods will usually have a measure of error (or accuracy) and unsupervised methods will not. This is because a difference can be seen between the observed behavior and the expected resulting behavior data that is present in the dataset- this information is simply not available to unsupervised methods.

Unsupervised learning can be said to learn from its interactions with the environment, where it knows the target state for the environment. Supervised learning can be compared to utilizing the aid of a teacher who is able to correct the network when an error is made.

2.3.3 Measures of Error

There are several mathematical ways to determine the accuracy of neural networks, which are described below. Although the methods can be used for other learning applications, focus is on application to neural networks. The terminology in each equation is provided in Table 2.1.

Symbol	*Meaning*
d	Resultant measure of distance, error or accuracy
$\vec{x}, \vec{y}$	Vectors which are being compared
x_i	The ith input pattern
y_i	The ith output pattern
n	Number of elements per vector. Vectors are required to have the same length.

Table 2.1: Symbol description for error measurement algorithms

Euclidian Distance

Euclidian distance measures a straight line distance between two exemplars, and is one of the most well known distance measures. Its function is as shown in Equation 2.4.

$$d(\vec{x}, \vec{y}) = \sqrt{\sum_{i=0}^{n} (x_i - y_i)^2}$$

Equation 2.4: Euclidian distance

Squared Error

The squared error has several variants:

- Mean Squared Error (MSE) takes the squared error and finds an average. When MSE is applied to a series of points, it produces a smoother error surface. Equation 2.5 indicates the formula used to calculate MSE.

$$d(\vec{x}, \vec{y}) = \frac{1}{n} \sum_{i=1}^{n} (x_i - y_i)^2$$

Equation 2.5: Mean Squared Error equation

- The Root Mean Squared Error (RMS, or RMSE) equation produces the square root of the mean squared error. The RMS error is not used as

frequently in applications in comparison to MSE, mainly because of the computational cost attributed to the mathematical square root operation.

- Normalized Mean Squared Error (NMSE) is a normalized version of the mean squared error. The NMSE takes into account the variance of the dataset to give a better fit to the reported error.

2.3.4 Classification versus Regression

Thought must be applied with respect to the desired behavior of the network, depending on whether it is required to predict the class of the input (classification), or required to predict continuous values of a pattern (regression).

Classification

Classification networks, in general terms, learn to predict a class value associated with its inputs. Classifiers can also be applied in such a way that a confidence factor can be returned in place of, or additionally to, a class value. An easy way to achieve this behavior is to separate the input data with respect to classes and present the data as 'is a specified class' or 'is not a specified class'. In this manner the network will learn that each input pattern is or isn't the specified class, functioning as a binary classifier. Separate binary classifiers can be implemented in parallel to learn other combinations of the input data.

Classifiers usually need to deal with input data that is not necessarily numerical and are more susceptible to receiving incomplete data.

Regression

Regression algorithms learn to predict an unknown function, when supplied with inputs to the function. This is why they are also known as function approximators, or interpolators. Application of regression models on a network

leads to the output of continuous-valued attributes, rather than the classifications or confidence measures associated with classifier learning.

Time series datasets are a variation on normal regression. Time series data sets are essentially chronological observations of a variable over time. This differs from standard regression datasets where if a variable is observed over time, the time itself needs to be somehow encoded into input- whereas in time series sets it can be implied. Time series sets are always specified in order of increasing time, and do not have any missing values (as the time index is implicit.) Hence, should a particular pattern be missed, the absence of the pattern will affect the datasets integrity. For example, if a variable is observed once a day for six weeks and an observation is missed, there will be (6*7)-1 patterns in the dataset, meaning that from the exempted point onwards the data for the following day will be encoded as the current day.

2.3.5 Extensions

There are many ways one can improve or vary the function of a network. Below are listed three commonly seen extensions to network algorithms that may have desirable effects on the network. However, it should be noted that implementation of any of the following techniques will not be automatically beneficial. Care must always be taken when choosing any parameter of an algorithm.

Weight Decay

Weight decay (also known as aging) can be employed in environments where units become less useful to the network over time. There are many ways to go about this task but the basic few are outlined.

- Decay all weights by a predefined amount on each training step

- Associate an age with weights, and apply a function to this age to find a decay term for the weight

- Associate a 'last used' or relevance term for each weight, to determine how relevant the weight is to the network's overall operation. There are many applications of this technique. For instance, when all weights' relevancies are known, the network is able to remove the n worst weights or all weights with relevance beneath a specified threshold.

One case in which weight decay may be opted for, is when weights assume a high relevancy when initially learning a problem, but lead to a too general solution. After the inclusion of the weight, other weights will adapt to refine the output, thus rendering the high-valued weight effectively useless.

Prematurely stopping training

When training a network, one may rationalize that more presentations of the data set is in fact better than fewer. Repeated observations of the dataset can make the network lose generalization ability by over fitting data. Three well known ways to determine when to stop training are at a pattern limit, an achieved accuracy, or a period of inactivity.

Processing time can be saved by stopping training when either a predefined number of epochs or training patterns have been seen. Careful selection of early termination criteria must be made; otherwise the network may not see all training examples, (making it naïve on some data) or not be affected by it (for example, if training is stopped after 50 pattern presentations when the dataset contains 3000 patterns).

Forced Observation

Forced observation can be employed in algorithms that are seen to add units (or otherwise adapt the network) without much consideration of the consequences

of doing so. In these cases the network can forcibly stop the addition of new units, such that the weights are able to train. This allows a period of settling to occur for several input presentations (or epochs) after a new unit has been installed into the network.

The Cascade Correlation (Fahlman and Lebiere 1990) algorithm employs this technique both at the conception of the network, and after adding a new hidden unit. It is necessary in this architecture as when a new unit is added, its input weights are frozen forever. For this reason, the making of rash decisions can have a serious detrimental effect on the resulting network, so forcing the weights to stabilize, may prevent the proposed unit from being installed.

2.4 Radial Basis Function Networks

Radial Basis Function (RBF) (Moody and Darken 1989) networks are a particular kind of neural network that are able to perform linear separation on normally non linear separable problems. RBF networks leverage the power of locally responsive units by using them for each unit in the hidden layer. Each unit is fully connected with each of the input and output units. RBF networks are typically only a simple three layer architecture (input- hidden- output). Any nodes that are installed into the network are placed within the hidden layer(s), and are linked fully with both forward and backward layers (output and input layers respectively).

The popularity of RBF networks is largely due to the fact that it makes use of Cover's theorem (Cover 1965), which states that non linear separable problems can always be separated linearly in higher dimensions. This is why the input is translated into the transfer space before it is processed. It also becomes possible to use other more complex activation functions like thin plate splines (Green and Silverman 1994).

2.4.1 RBF Training

A general RBF training algorithm can be stated with the following pseudocode:

1. The network is initialised with a predefined number of hidden units. One node is created for each input, and another for each output. The input and output layers are fully connected at this stage. Units are randomly placed throughout the transfer space.

2. Present the selected example to the network.

3. If the overall network performance is acceptable then cease training. Acceptability is measured by comparing the networks response to the desired response and applying a threshold comparison to it.

4. Update the network to be more suited to the current input pattern. This can involve moving the units around the transfer space, and by modifying the peak value of any number of units.

The main area of research with respect to RBF networks is in the centroid placement (de Castro and Zueban 2001; Fritzke 1994). Other less active areas are generalization ability and pruning. Basis functions can have any functional mapping, but the Gaussian mapping is by far the most popular.

The term 'locally tuned' is usually associated with RBF style networks. It refers to the way that a node responds to stimulus, that is to use a distance measure between a node and the input in hidden unit space.

2.5 Resource Allocating Network

The Resource Allocating Network (Platt 1991) takes a constructivist approach to the aforementioned RBF network. It was created in an effort to achieve a more compact network representation over statically sized RBF networks for function

interpolation problems. The focus of this thesis is to apply competitive learning techniques to this Resource Allocating Network, such that the underlying RAN implementation will be used in every experiment run.

2.5.1 Training of the RAN

The process of training the RAN is quite similar to the training of RBF networks.

1. The network is initialized with no hidden units, and a bias unit is connected to the output layer units.
2. An input pattern is presented to the network.
3. The overall response from the network on the input pattern is measured
4. Each hidden units output is measured, such that the closest units distance to the input is stored.
5. The closest centers location is compared to a minimum distance of interest measure, if it is exceeded and the network error value exceed a predefined limit, add a new unit
    a. The new unit will be centered to suit the current pattern being presented.
    b. Should this be the first unit to be added to the network, its width will be set to the product of the constant width value and the current distance of interest value, otherwise the width will be set to the product of the constant width value and the distance to the closest centre to the input pattern.

6. If a unit is not added in step 5:

    a. Use the Widrow-Hoff LMS algorithm (Widrow and Hoff 1960) to minimize the output layer weights shown in Equation 2.6.

$$\Delta\vec{h}_j = \alpha(\vec{T} - \vec{y})x_j$$

Equation 2.6: Hidden to output unit weight update formula

    b. Use the Widrow-Hoff LMS algorithm to minimize the error of the bias unit shown in Equation 2.7.

$$\Delta\vec{\gamma}_j = \alpha(\vec{T} - \vec{y})$$

Equation 2.7: Bias weight update formula

    c. Adjust the centers of the units, according to Equation 2.8.

$$\Delta c_{jk} = 2\frac{\alpha}{w_j}(I_k - c_{jk})x_j\left[(\vec{T} - \vec{y})\cdot\vec{h}_j\right]$$

Equation 2.8: RAN centre adjustment formula

Reference can be made to Table 2.2 for the definition terminology used in these equations.

Variable	*Meaning*
$\vec{h}_j$	Output layer weight j
α	Learning rate
$\vec{T}$	Expected output pattern
$\vec{y}$	Network response
x_j	Synaptic strength of hidden unit j
$\vec{\gamma}_j$	Bias response for output j
c_{jk}	Center for unit j for input dimension k
w_j	Width of hidden unit j
I_k	Input k

Table 2.2: Description of RAN training parameters

2.6 Competitive Learning

Competitive learning is a process where units must compete for the ability to adapt to input. The goals of competitive learning are similar to that of other machine learning techniques, that is: to minimize error, maximize population diversity, and to detect features of the input (Fritzke 1997). Competitive learning can be used for density estimation, clustering, and visualization applications.

Of most concern in this study is application of competitive learning for training centers of a Resource Allocating Network.

2.6.1 Hard Competitive Learning

Hard Competitive Learning (HCL) is a process where units vie for input signals. Each input can only be 'won' by one unit, giving the winner the exclusive right to adapt. The winner's exclusivity of adaptation is why the method is also known as 'winner-take-all learning'.

Initial placements of units can have a large effect on the network. The winner take all approach of this type of learning can lead to the death of units. Dead units can be defined as:

> "These are units which – perhaps due to inappropriate initialization – are never [the] winner for an input signal and, therefore, keep their position indefinitely." (Fritzke 1997, p. 10)

Fritzke also suggests that one should treat dead units as harmful because they do nothing to accomplish the goal of the network, other than needlessly consume resources. The problem of dead units can be minimized by choosing initial unit placements based upon probability of input signals. By selecting more common input features, the units are more likely to win over their rivals because there will be fewer of them to compete, and they will have a greater degree of separation. However, this required prior examination of the training data.

The 'select units based on probability' method will select more popular input combinations more often, leading to the creation of more densely populated areas of input space. Fritzke states that this situation may be suboptimal for some tasks, commenting that it may be a better idea to take an inverse approach of adding fewer units in the more popular input space, and adding more units in the more sparsely populated input space (Fritzke 1997).

Hard competitive learning can be undertaken in both online and offline environments. Offline methods focus on processing all input signals before any adaptations are made to the structure, whereas online methods are able to adapt to individually presented input signals. By definition, online techniques cannot be applied to data that comes in a stream format, or does not have a finite size.

k-means

The k-means algorithm is an example of a hard competitive learning algorithm (MacQueen 1967). The k-means algorithm has an individual learning rate for each unit in the network. The learning rate is set by the function in Equation 2.9, where the t parameter refers to how many input signals the particular unit has been the successful competitor. The more times that a unit has won a competition is directly related with how far it is willing to move, as it assumes by winning it is only getting more suited to the input pattern being presented.

$$\alpha = \frac{1}{t}$$

Equation 2.9: The k-means learning rate function

2.6.2 Soft Competitive Learning

Soft competitive learning is the same as hard competitive learning, with the exception that the exclusivity of winning neurons is reduced and shared around multiple winners. Soft competitive learning reduces the risk of units dying, as units are more likely to receive some form of reward, since the reward is distributed to all the runners-up of the competition. A distinction between soft competitive learning with fixed network dimensionality, and without fixed network dimensionality can be made (Fritzke 1997). The case of fixed network dimensionality will not be discussed here as the focus is on constructive network architectures.

Neural gas

The neural gas (NG) algorithm is an efficient competitive learning algorithm (Martinetz and Schulten 1991). The neurons in the network are arranged with respect to the Euclidean distance between their weight and input vectors.

The NG algorithm was adapted in Bogacz and Giraud-Carrier's 1995 paper to improve the selection of the position of the centroids, and attempts to address the issue of non optimal network configuration when using unsupervised learning (Bogacz and Giraud-Carrier 1995). The algorithm attempts to install and move network units toward heterogeneous clusters by adding new neurons for subclasses of data. Neurons are said to be attracted more strongly by input of its currently predicted class (hence belonging to its internal data set), and more weakly for data that is not in its internal set.

Pseudocode for the Neural Gas algorithm is as follows

1. Present the network with a selected input pattern ξ
2. Sort the network's units by their distance to the selected input pattern
    a. Denote w_{i_0} as the closest units response to the input, w_{i_1} as the second closest unit, to the k[th] far away unit w_{i_k}
3. Update the units' centers with Equation 2.10. The term $k_i(\xi, A)$ is said to represent the rank k of the unit A on an input ξ.

$$\Delta w_i = \left(\varepsilon_i \left(\frac{\varepsilon_f}{\varepsilon_i} \right)^{\frac{t}{t\max}} \right) \cdot \exp\left(\frac{-k_i(\xi, A)}{\left(\lambda_i \left(\frac{\lambda_f}{\lambda i} \right)^{\frac{t}{t\max}} \right)} \right) \cdot (\xi - w_i)$$

Equation 2.10: Neural gas centre adaptation formula

4. Increase the value of t (the number of inputs it has seen)
5. While the value of t is smaller than tmax, continue learning, otherwise halt.

Growing Neural Gas

The growing neural gas algorithm (Fritzke 1995) is another extension to the neural gas algorithm. It is an extension from his previous work that also incorporates topology information (Fritzke 1992).

Rival penalized Learning

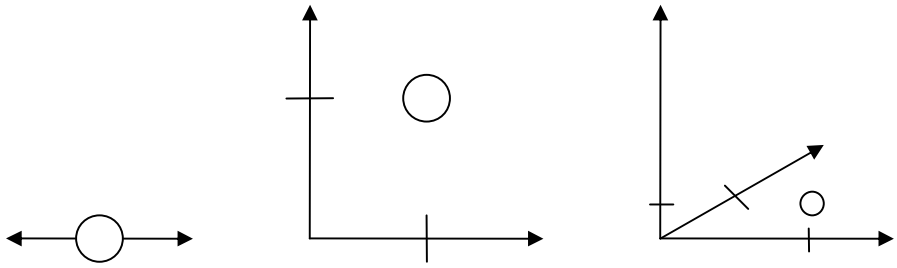
Another type of competitive learning is rival penalized learning. Rival Penalized learning has a single winner selected for adaptation, much like hard competitive learning but also pushes its competitors away from the input location, effectively making competitors de-learn the input.

2.7 Benchmarks

2.7.1 Input

Dimensionality

The 'curse of dimensionality' refers to the fact that each attribute in a dataset exists in its own dimension when examined from a global point of view (Bellman 1961). Figure 2.2 depicts a single data point that has one, two or three attributes. Although the data becomes increasingly difficult to visualize, more data points are required to map each dimension accurately with respect to other dimensions.



(a) One dimension (b) Two dimensions (c) Three dimensions

Figure 2.2: Pattern Complexity for 1, 2 and 3 attribute data

Attribute (in)dependence

Having independent attributes makes learning a given dataset simpler, as each attribute can be learnt in isolation from other attributes. There are, however cases in which attributes are dependent on other attributes. In these cases the network cannot effectively learn the dataset if it examines each dimension by itself.

2.7.2 Test and Training Datasets

When analysing these techniques it is necessary to split the data into test and training sets. Splitting of data in this way enables the network to be trained on data that it has access to- simulating a production environment where future information simply is not available. Once trained, the network is can be quizzed and subsequently assessed on the how it reacts to unknown cases. A network is usually put into a 'read only' mode when being presented with an item of a test dataset, whereby its network characteristics are fixed. By stopping an algorithm from learning data in the test set, the test set is effectively still unknown to the network, allowing for the test set to be presented at a future time

3 Methodology

This section is intended to be a roadmap to the implementation of the work that was used to gather the results presented in the following sections. This section endeavors to detail the following

- Parametric breakdown of the RAN training procedure
- Choice of competitive learning algorithms, and their parameters for their learning
- Choice of datasets, how they differ and why they were selected for use in this work
- The testing procedure, showing what variables were varied in which tests such that coherent results were able to be produced

3.1 Base RAN Implementation

3.1.1 Upper limit of network size

In the implementation, network size has been constrained to 5,000 hidden units. For reasons of speed, the base RAN implementation utilises statically dimensioned arrays (instead of variable length, more computationally complex alternatives) to hold unit information. This trade-off requires a maximum hidden layer size to be selected. The figure of 5,000 was arbitrarily chosen to account for running of the software on limited memory machines and to handle unexpectedly large networks. This limit was reached on several occasions when the Building dataset was being used, but was not reached when being trained on any other datasets.

3.1.2 Bias Initialisation

Platt's paper made no mention as to what the bias synapse strengths should be initialised to (Platt 1991). It was decided to initialise the weight to a random value between -0.5 and 0.5.

3.1.3 Distance Decay

The decay constant τ used in equation 2.9 of Platt's paper was undefined. However, the value of τ was calculated by using limits specified elsewhere in the paper (Platt 1991). Equation 3.1 indicates the formula used to calculate τ. And for this equation, 'time' is the number of input presentations at which $\delta_{\min}$ is reached, and intermediate values of this decay function are stored in δ.

$$\tau = \frac{time}{\ln\left|\frac{\delta_{\max}}{\delta_{\min}}\right|}$$

Equation 3.1: Calculation for Tau

$$\delta(t) = \max\left(\delta_{\max}\ \exp\left(\frac{-t}{\tau}\right), \delta_{\min}\right)$$

Equation 3.2: RAN distance decay formula

3.1.4 Default RAN Parameters

The parameters described in Table 3.1 are the parameters used in RAN-trained experiments. Additionally, these parameters are inherited into the other training algorithms, unless otherwise specified. These parameters have been constructed from those referred to Platt in his 1991 paper (Platt 1991).

Parameter	*Value*	*Comment*
$\delta_{\max}$	0.7	Initial value of decay parameter sigma
$\delta_{\min}$	0.07	Final value of decay parameter sigma

$\delta(t)$	Calculated	See Equation 3.3
κ	0.87	Multiplier of minimum distance of interest measure when allocating a new unit
ΔT	85	Specifies which output pattern is expected (i.e. $t + \Delta T$)

Table 3.1: Default RAN parameters

3.2 Extensions

After verification of the results produced by the underlying network, the program was extended in its abilities, such that it could be trained by other methods.

3.2.1 Neural Gas

The neural gas algorithm was selected as a soft competitive training strategy as it is a well known, well performing algorithm (Bogacz and Giraud-Carrier 1995). It was intended that the use of this algorithm could influence the direction to go with respect to which future soft competitive learning strategies would be selected. The actual code used was based heavily on that used in (Loos 1998) and as such, many of the parameters are the same, as shown in Table 3.2

Neural Gas Parameters		
Parameter	*Value*	*Comment*
EI	0.5	Initial value of epsilon
EF	0.005	Final value of epsilon
ET	0	Value of epsilon(t)
LI	0	Initial value of lambda
LF	30	Final value of lambda
EPSILON	0.1	Value of epsilon
MAX_TIME	The product of the training points in dataset and the amount of training epochs	Time at which movement reaches a minimum

Table 3.2: Parameters used in the Neural Gas algorithm

3.2.2 Hard Competitive Learning

Two kinds of simple HCL algorithm were considered to be applied to the datasets.

Non stationary input

This HCL variant was able to take into account moving data points. The concept of moving data points can be envisaged by attempting to learn the inner coordinates of a bouncing box in a 2 dimensional environment. This method was initially implemented as it was thought that the ability to take into account moving data may result in the discovery of 'windows' in the dataset. For these windows, it was expected to obtain a more compact and lower-error window. This approach to the data was trialed and compared to the stationary input technique, and it was found to be on average poorer in terms of network size

and error. As a direct consequence, no final results were calculated using this variant.

Stationary Input

The other approach considered to standard hard competitive learning was unable to take into account the mobility of the data points as the non-stationary technique does. This variant of the original HCL was implemented after some review of the initial results, which indicated a similar network size to the networks trained by the neural gas algorithm had developed.

3.3 Datasets Used

3.3.1 Process of selection

Two main sources of information were considered when selecting which datasets to use. It was deemed that only a limited number of datasets should be chosen, to simplify the experimentation procedure and amount of raw data to consider. One source of information was Platt's 1991 paper , in which the Mackey-Glass dataset was used to showcase performance of the Resource Allocating Network (Platt 1991). Another major source that was used to compare potential datasets was the Proben1 database (Prechelt 1994).

It was decided to keep the Mackey-Glass time series dataset such that the competitively trained networks could be compared back to the initial Resource Allocating Network. Due to the way the RAN was constructed in order to validate the implementation, only regression datasets were chosen. After consideration was made to dataset size, the Flare and Building datasets were selected.

3.3.2 Dataset usage

At the start of each experiment datasets are split into testing and training sets. Training sets will only ever be presented to the network whilst training, and the test set is used to ascertain network performance on unseen data. Unless otherwise specified, the final 10% of a dataset when loaded from a file is identified as the test set, with the remaining 90% used for training.

Before presenting each data set to the network for each learning epoch, the data was shuffled at random. This action stops the networks from learning relationships that are based on any implicit ordering inside the dataset, either known or unknown, which has great potential to disrupt the incremental construction of the network. For instance, the inherent ordering of time series data is based on time.

3.3.3 Description of Datasets

Mackey-Glass time series

The Mackey-Glass data set is a mathematical representation to the rate of production of white blood cells regulated by biological feedback systems (Mackey and Glass 1977).

The dataset used consists of 120,000 data points, and was generated from source code provided by Roger Jang (Jang 1992).

The training set generated consists of 500 points starting from t=0, and the test set was 500 points starting at t=4000, in accordance with those used by Platt (Platt 1991). The structure of the Mackey-Glass dataset is shown in Table 3.3, and the parameters for generating the set can be found in Table 3.4.

Index	*Label*	*Attribute Type*	*Min – Max*	*Standard Deviation ± Error*
First, but ignored	Time Index	Discrete	0-120000	2.89E+02 ± 9.13E+00
Output 1	Y Value	Continuous	1.31E+00 – 2.19E-01	2.37E-01 ± 7.49E-03

Table 3.3: Mackey-Glass time series description

Variable Name	*Value*	*Comment*
step_size	0.01	Size of integration step
x	-0.6	Initial condition
sample_n	1200000	Total number of samples to take, not including the initial value
tau	17	Delay constant
interval	100	Write value to screen every interval steps
time	-1000	Time at which to start

Table 3.4: Parameters for the Mackey-Glass generation program

Flare

The dataset 'flare' (Prechelt 1994) that is used is a variant of the UCI-provided flare dataset (Newman et al. 1998). The choice of the modified flare dataset was justified by its inclusion in the Proben1 dataset (Prechelt 1994). The modifications changed the first three UCI-Flare parameters into 7, 6 and 4 binary values respectively, allowing the dataset to be fully represented by values in the 0.0-1.0 range. The set describes the conditions surrounding electromagnetic explosions from the sun; that have the propensity to create large scale disruptions with power distribution facilities, and produce the effect of auroras.

The set consists of 1066 patterns. The training set consists of the first 867 patterns, and the test set consists of the remaining 199 patterns. A description of the dataset can be found in Table B. 1.

Building

The Building dataset (ASHRAE 1993) was designed for a competition designed to test prediction algorithms. The dataset models energy consumption of a building, based on external environmental effects. It has 11 attributes: 8 inputs and 3 outputs. Four of the 8 inputs are used to encode the time at which the readings were taken: month, day, year and 4-digit hour & minute- all of which are discrete valued. Table B. 2 describes the structure of the dataset.

The dataset covers the period of time between the 1st of September 1989 and 22nd February 1990, a total of 4182 patterns. The training set consists of the first 3685 patterns (1 September 1989 to 3pm 1st February 1990), and the test set consists of the remaining 497 patterns (4pm 1st February 1990 to 9am 22nd February 1990). A description of the dataset can be found in Table B. 2.

3.4 Experiment Notes

3.4.1 Definition of Available Parameters

The results presented in the 'Results and Discussion' section were found by running a series of experiments. An experiment is specified in terms of its learning rate, dataset, center movement strategy, and how many epochs it was trained for. The actual parameters available to each experiment are defined in Table C. 1. The first four parameters to the program specified are required to be specified in the order shown in the table, and do not need to be prefixed with the parameter name. For example the first argument expected is a value from the dataset row, the second argument expected is a value from the movement

strategy row, etc. The remaining parameters are optional and are used in the format "parameter value".

3.4.2 Program Output

For each experiment run, several files are created. These files store all the result information that has been collected over the course of the experiment. They include:

- The total number of hidden units in the network
- The number hidden nodes in the network at each training epoch
- The time taken to run each iteration in milliseconds
- The average test and training errors of each final network
- The average test and training error at each training epoch

A program was also been created to collate this information into a tabulated form for easy addition into a spreadsheet.

Measurement of Time

Iteration time was calculated by polling the system clock at the start and end of each experiment iteration and finding the difference between the two figures. This is accomplished by the System.getCurrentTimeMillis() java method. It is understood that this method of calculating runtime is poorer than some other methods available, but was deemed acceptable as each experiment was run 20 times in order to gain a representative average, and an error of a few seconds represented a very small percentage of the total runtime. Each experiment was run on an identically configured 2.8 GHz Pentium 4. Each machine therefore was able to get the same sized amount of work done in the same amount of time, making it possible to collect data from multiple computers as though they

were a single unit. By using machines of the same specification and platform, the differences between average times to do a measure of work is decreased considerably allowing a larger volume of data to be collected.

3.4.3 Training Schemes

Three individual training schemes were used. In all cases, the competitive training schemes modified the center placement of the units in the hidden layer and nothing else. The three main strategies used were

- Default RAN center allocation
- Neural gas soft competitive learning
- Hard competitive learning (using the stationary data algorithm)

3.4.4 Training Epochs

Each experiment was run four times (one each of 100, 200, 400 or 800 training epochs) in order to be able to demonstrate the performance and accuracy of each network. It is important to note that four individual runs of the networks were done, as opposed to an 800 training epoch run segmented into 100, 200, 400 and 800 sets.

3.4.5 Learning Rate

For each experiment, two learning rates were trialed. One was alpha=0.05, and the other was alpha=0.02. One feature of the code used was that a different learning rate was able to be used for the hidden and the output layer weights, but in all cases, both hidden and output learning rates were set to the same value.

3.4.6 Figure Labels

Figures provided have their own notation. Most series names can be deconstructed by reading the components of the name.

- RAN/HCL/GAS refers to the training method used. That is, Default trained RAN, Hard Competitive Learning, Neural Gas
- 100/200/400/800 refer to the maximum training epochs that the series used
- Glass/Flare/Building refers to the dataset that was used. That is, Mackey-Glass, Flare or Building

3.4.7 Result Stability

Each experiment has been run 20 times unless otherwise stated in Table 3.5. This figure of 20 was arbitrarily chosen to be a compromise between speed (a small number of runs) and a decent statistical average (gained by a large number of runs). Table 3.5 shows that the Building dataset was under-sampled for several experiments. This was primarily due to the long period of time taken to run the experiments.

Training Scheme	*Dataset*	*Epochs*	*Learning Rate*	*Trials Run*
RAN	Building	200	0.05	10
RAN	Building	200	0.02	10
RAN	Building	800	0.02	10
HCL	Building	100	0.05	10
HCL	Building	200	0.05	10
HCL	Building	400	0.05	10
HCL	Building	800	0.05	10
HCL	Building	100	0.02	10

HCL	Building	200	0.02	10
HCL	Building	400	0.02	10
HCL	Building	800	0.02	10
GAS	Building	100	0.05	10
GAS	Building	200	0.05	10

Table 3.5: Trials run for experiments not trialled 20 times

3.4.8 Confidence Factor

When graphing results, two extra points were added for every data point. The extra points indicate the confidence in the actual data point, which can be thought of as an alternative to a standard error in application. These points were calculated this using Equation 3.3. α is always fixed to the value of 0.05, to give a 95% confidence in the factor. σ is the standard deviation of the entire source data, and n represents how many items there are in the source data. The calculated confidence factor is then added and subtracted from the average value, to give an impression as to the real nature of the data, indicating if the average has been skewed by outliers. Graphs that are presented in the following section may also include a confidence-factor version in Appendix A.

$$confidence = \bar{x} \pm (1-\alpha)\left(\frac{\sigma}{\sqrt{n}}\right)$$

Equation 3.3: Equation that is used to calculate the confidence intervals

4 Results and Discussion

As there are multiple hypotheses that were tested, this section presenting the results found has been split. The first subsection will be aimed at the results that have some focus to the resultant network size, and the latter subsection will be aimed at examining resultant network accuracy.

4.1 Network Size

This section of results is dedicated to examining the result data with respect to the following hypothesis:

> The use of competitive learning to train the centers of Resource Allocating Networks will reduce the number of hidden neurons required in comparison to the default behavior of the Resource Allocating Network.

4.1.1 Size comparison with respect to training epochs

Figure 4.1 depicts network size of the three RAN variants with respect to the number of epochs they have been trained over. More scope to this figure is given in Figure 4.2, where network size is shown for experiments trained for 100, 200, 400 and 800 epochs. Both the default and the neural gas trained RANs appear to have the same shape graph, whereas the HCL trained network has a flatter, slower growth rate than the two other types of network. When the data is presented in the extended format in Figure 4.2, the separation between all three variant networks becomes clearer, with the default-trained RAN size increasing without limit, and the neural gas trained RAN reaching a plateau slightly above that of the HCL trained version.

When examining Figure 4.1, it is clear to see that each of the variant's graphs have roughly the same shape before 20 training epochs. After these initial 20 training epochs, each network increases in size. However the neural gas

network achieves a smaller network size than the other two networks, although this doesn't persist over a longer trial.

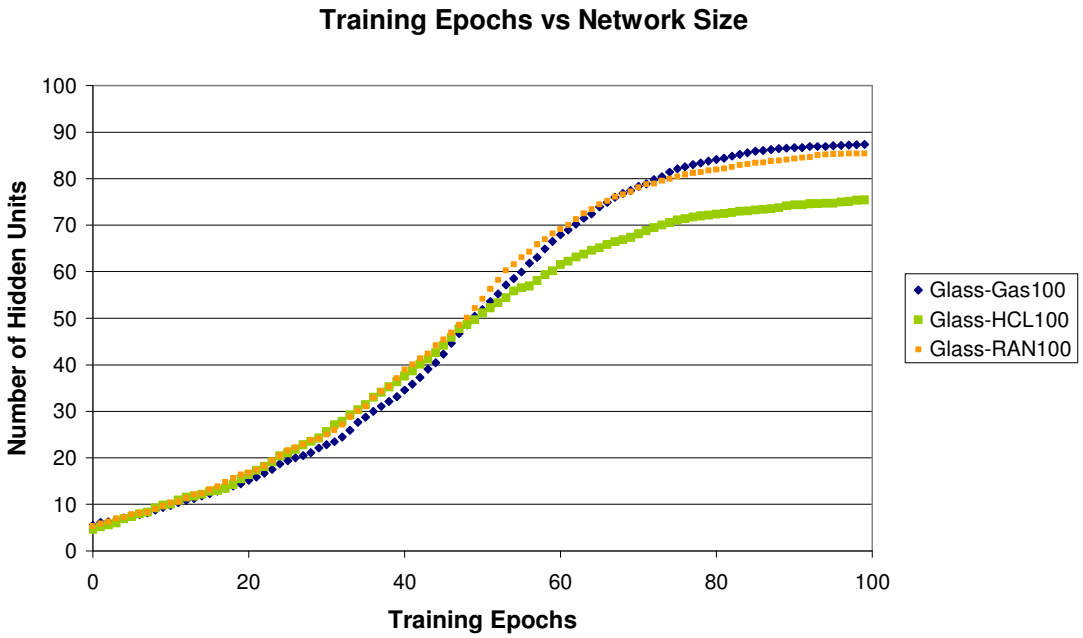


Figure 4.1: The resultant network size of architectures trained over 100 epochs

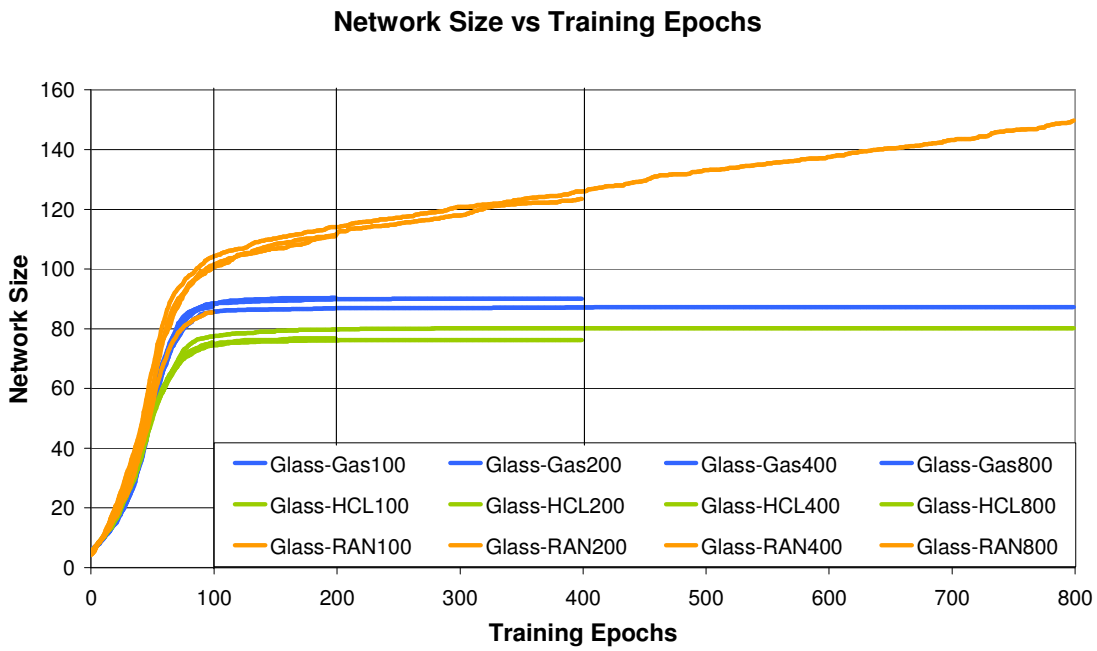


Figure 4.2: The network size of architectures trained over 100, 200, 400 and 800 epochs on the Mackey-Glass dataset

As shown in Figure 4.3, the Resource Allocating Network follows a tight initial growth curve, and after reaching around 90 hidden units, the rate of change (blue) of the network size drops. Unfortunately, the rate of change does not continue to fall past this point, yielding unbounded network growth. The confidence interval indicated in grey demonstrates that this growth is steady throughout all the trials as the curves follow the center line very well, having a maximum separation of about 10 units.

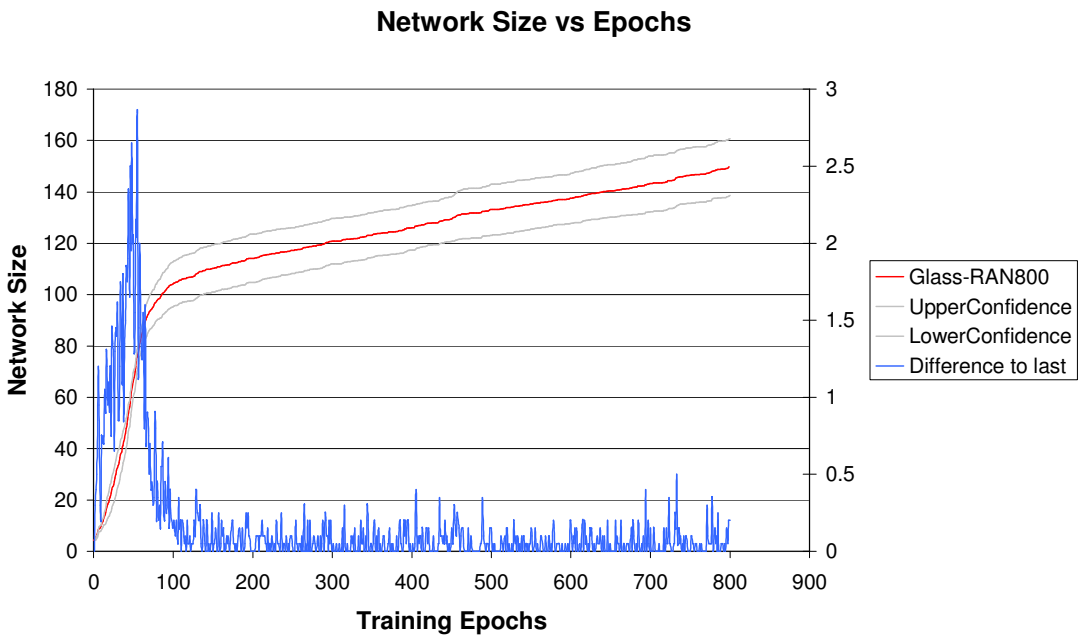


Figure 4.3: Resource Allocating Network size versus training epochs presented from the Mackey-Glass dataset

4.1.2 Network size comparison with respect to datasets

Figure 4.4, Figure 4.5 and Figure 4.6 depict performance in terms of network size versus how many passes over the training set they have had. In each figure, the orange series represents the network size for the Mackey-Glass equation, the green series represents the network size for the Flare dataset, and the orange series represents the network size for the Building dataset. Two trends are visible by comparison of the graphs.

Graph Size

The first is that there is a distinct order in terms of size of the dataset. The resultant size of a network trained by Mackey-Glass is the lowest series in each graph. Also, for each figure, the Flare dataset always produces networks that are slightly larger than the Mackey-Glass trained networks. The building dataset always produces by far the biggest network.

Each dataset used produces similar results in terms of the overall shape of the network. The most prominent observation for this is the shape of the Building datasets, but it can also be seen at the start of each of the networks for the Mackey-Glass and Flare sets. Of the graphs, the RAN has the largest network size for each set, as 15 of the 20 RAN experiments trained on the Building dataset reached the maximum network size, whereas this limit was not reached in any other trial.

Position of Plateau

It can be observed from each figure the point at which each network reduces its unit addition rate. For each network, on each dataset, this limit is reached at around or before the 120 epoch mark, with the only exception being the RAN trained Building dataset. Confidence intervals have been applied to the Building results, in order to demonstrate how well the average figure shown represents the true nature of the data. For the RAN trained building data it indicates a very poor conformity with the difference being 970 units.

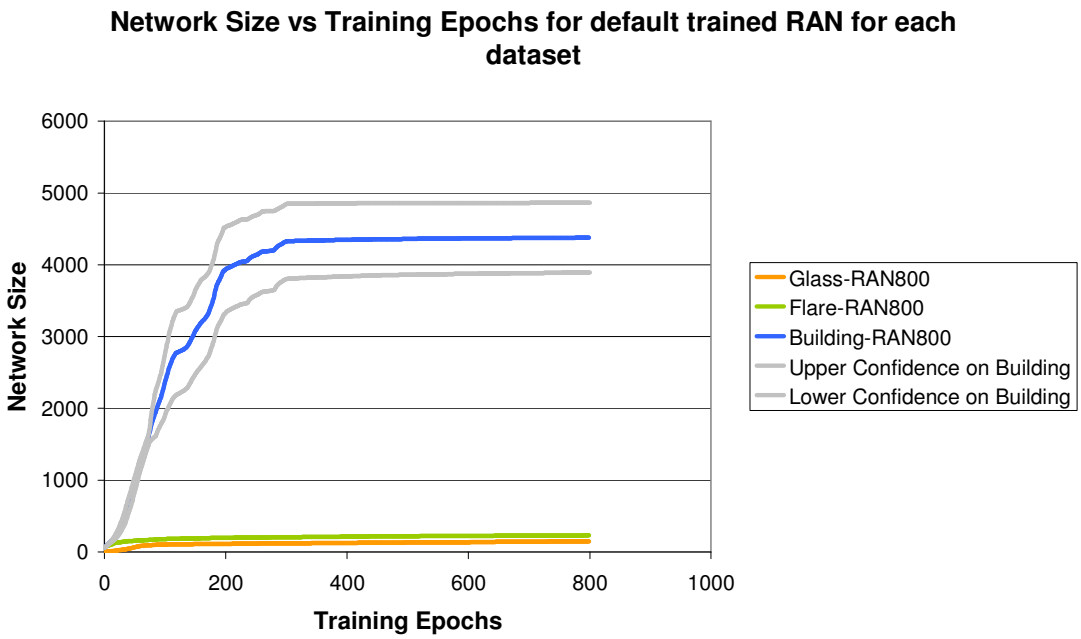


Figure 4.4: Size of default trained RAN on each dataset with respect to training epochs

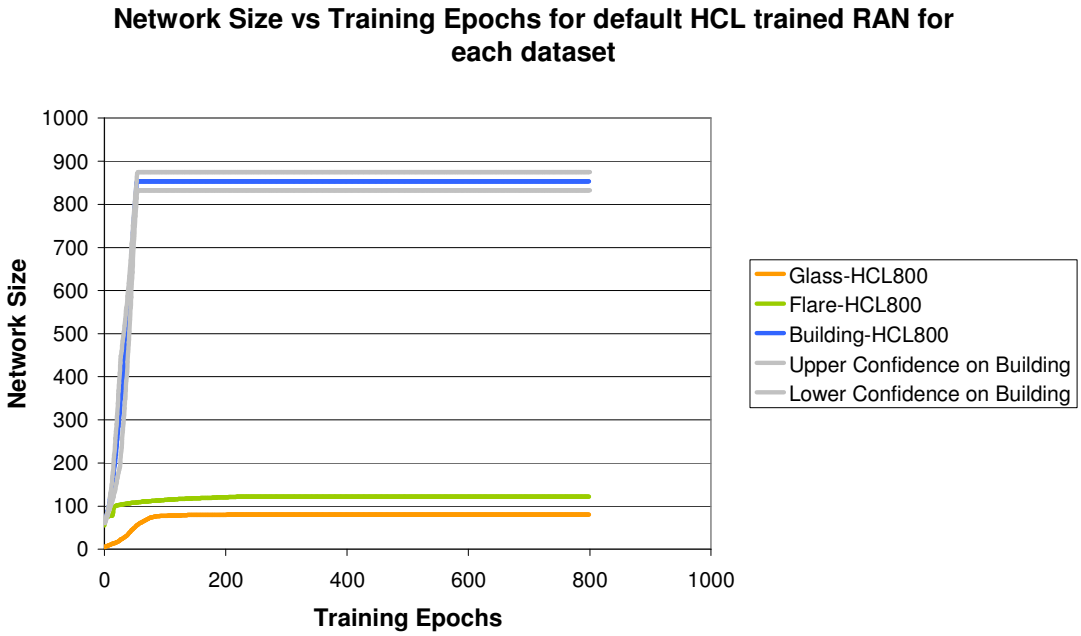


Figure 4.5: Size of HCL trained RAN on each dataset with respect to training epochs

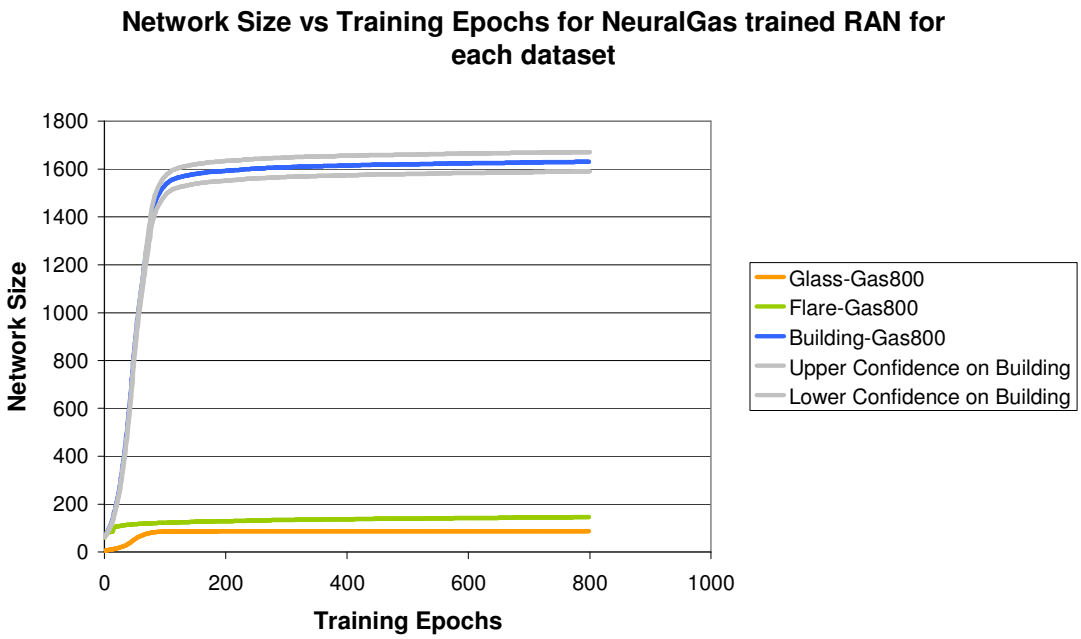


Figure 4.6: Size of Neural Gas trained RAN on each dataset with respect to training epochs

4.1.3 Importance of Initial Patterns

Examination of Figure 4.7 exhibits an upward trend in terms of network size in relation to the number of training epochs encountered. Though all three series end in the same trend upwards, there is a noticeable discrepancy between the 100 epoch trained RAN, and the 200 and 400 epoch trained RANs. One reason to explain this is that the Resource Allocating Network places a high importance on the first training points that are presented to it, giving its final size a degree of uncertainty. This behavior can be seen because of the way the network decays unit width with respect to units installed in to the network. However, when Figure 4.7 is compared to Figure 4.8, the same association cannot be made, even though the underlying incremental construction of the network is the same. This is likely due to the fact that the competitively trained networks have a better ability to shift their centres than the default behavior RAN. The RANs inability to limit network size comes from the described (see section 2.5.1) fixed-learning rate scheme used and thus, exhibits this behavior. This compares

favorably to the more adaptable competitively trained networks, which have a more dynamic learning behavior than the default trained RAN, making them less constrained in their movement. Meaning they can give a better rate of coverage over the training data, and a smaller network size.

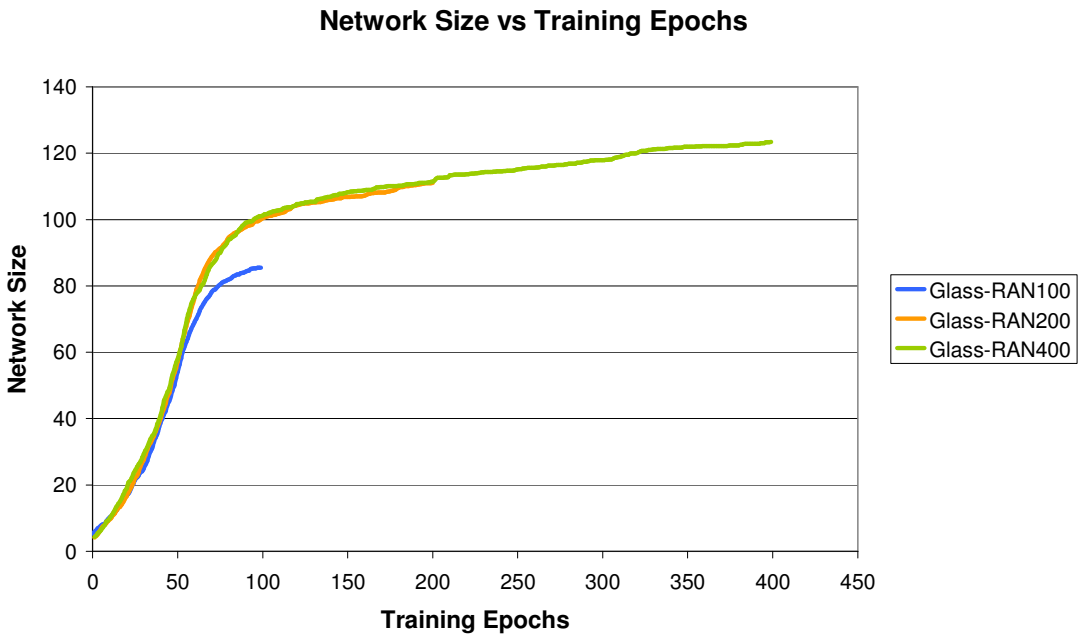


Figure 4.7: Comparison of the default Resource Allocating Networks over 100, 200 & 400 training epochs of the Mackey-Glass dataset

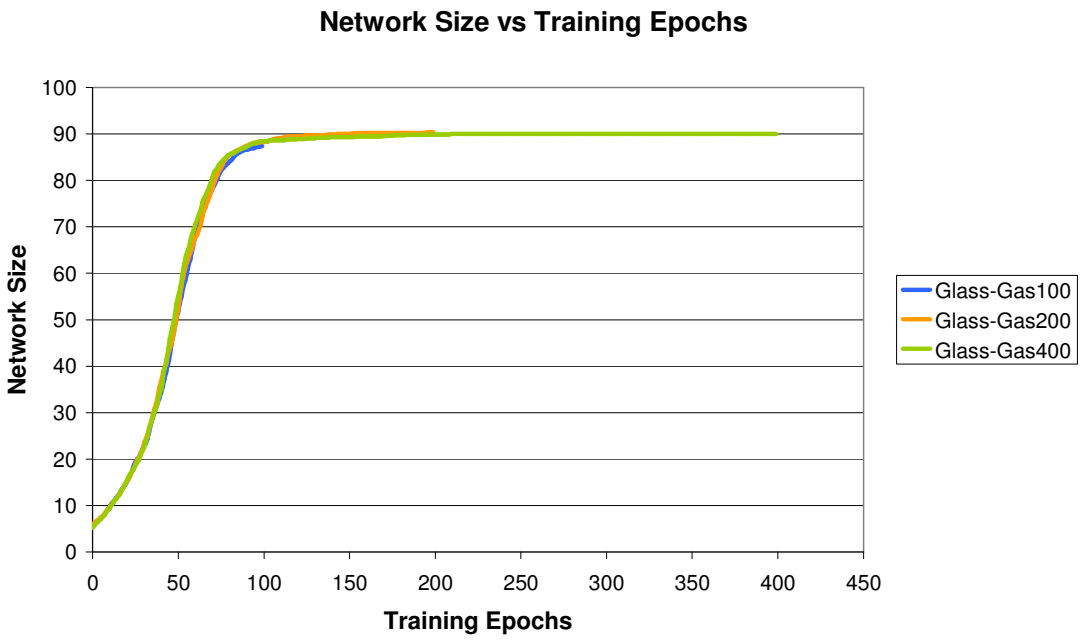


Figure 4.8: Comparison of neural gas networks trained over 100, 200 & 400 training epochs on the Mackey-Glass dataset

4.1.4 Limitation of Network Size

As can be seen in comparison of the RAN and CL trained networks (Figure 4.7, Figure 4.8, and Figure A. 2) the networks trained by the default RAN strategy never reach a size plateau within the iteration criteria used whilst experimenting. One explanation for this is that because the network adapts all the hidden layer units at each input presentation. This means that the units never end up being evenly spread so when they are trained they tend to be drawn together, ending up representing points that are already covered.

4.2 Training Speed

This section of results is dedicated to examining the result data with respect to the following hypothesis:

The use of competitive learning for training locally-responsive candidate neurons will increase training speed of a Resource Allocating Network.

There are two ways of measuring the training speed of a neural network. The first is to examine the rate of change in the observed error metric, and the other is to examine the actual runtime of the training process.

4.2.1 Training speed with respect to training epochs

Relationship between time and training epochs

Each training method shows an increase in training time that is close to linear with respect to how many epochs it has been trained for. The relationship comes from the close association that network size has with the amount of times it has been presented with a data set. As the amount of processing required is based on the underlying size of the network. Thus it can be reasoned that because the size of the network increases with some respect to the amount of training epochs, the amount of processing required will also increase. For instance, if a network were to double in size, the network must do twice the amount of calculations for each future pattern presented to it.

Training Efficiency

Figure 4.9 demonstrates training accuracy over each of the training strategies on the Mackey-Glass dataset. Each of the learning accuracies over the training period are intertwined with each other, with the exception of the first 50 epochs. Figure 4.10 is a zoomed in version of Figure 4.9, showing the first 100 training epochs of the 800 epochs that the experiments were trained over. This second graph suggests that for the first 50 training epochs the network is beginning to come to terms with the dataset, as the reported error decreases slightly when compared to the rate of change of the network size observed later in the training session. After this barrier of about 50 epochs, the reported error for each network follows a very similar downward trend, flattening out at around the 350 epoch mark. One point of commonality for training each of the networks is how the output layer units are trained. This may yield an explanation of the

strikingly similar error rates found, as the competition merely distributes the units more sparingly than the RAN does. Another plausible explanation of this is that the measure of error used for comparison may not have enough resolution to properly distinguish each of the series.

At the end of the 800 training epochs shown, the HCL trained network only just outperforms the default and neural gas trained networks. A relationship may be drawn in this example between network size and network accuracy with respect to the HCL trained network- a more compact representation allows the training to shift the networks units in a more efficient fashion. This could occur such that the units are sufficiently far apart that they will never be drawn together, effectively becoming wasted but still achieving a comparable error rate to the other networks. Unfortunately this argument does not hold when being compared to the default RAN trained network- even at the 350 epoch mark the network is still adding units, without decreasing its error.

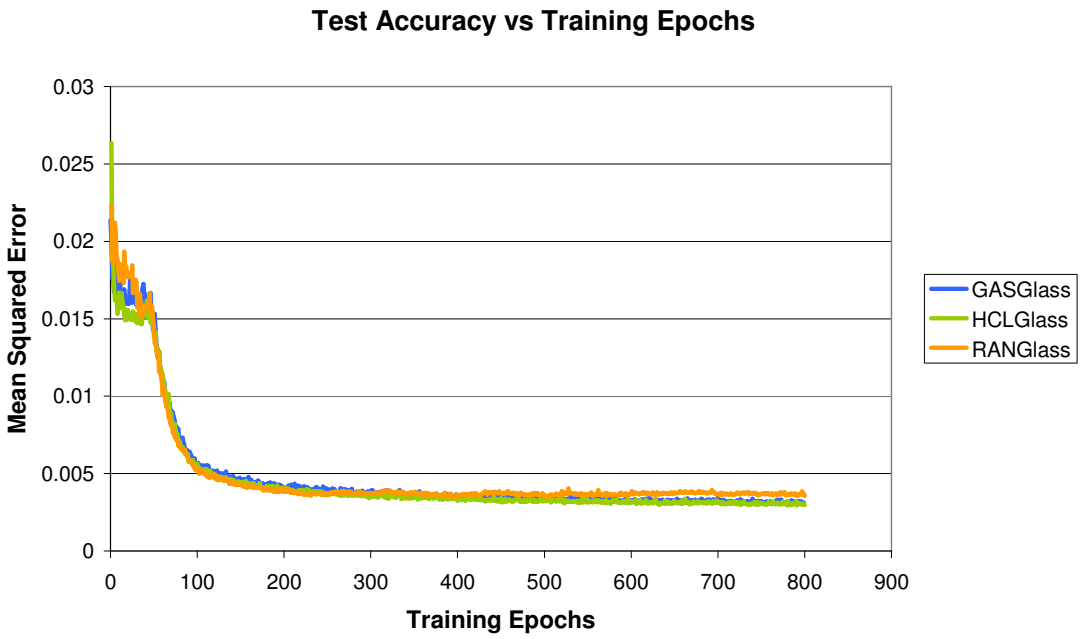


Figure 4.9: Average test accuracy for each training strategy

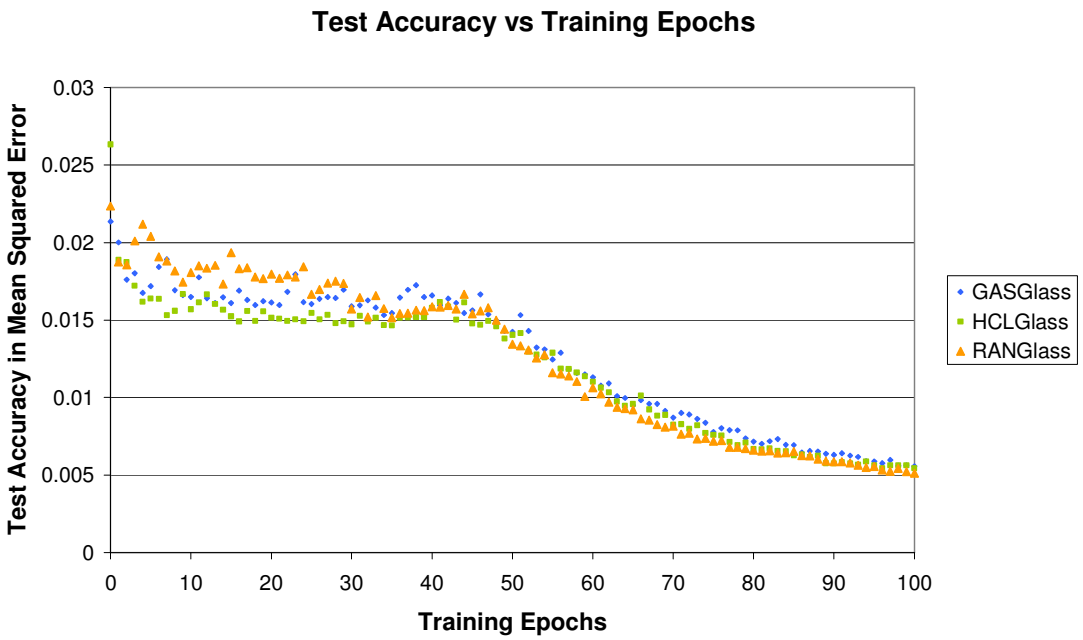


Figure 4.10: Average test accuracy for the first 100 epochs for 800-epoch trained experiments

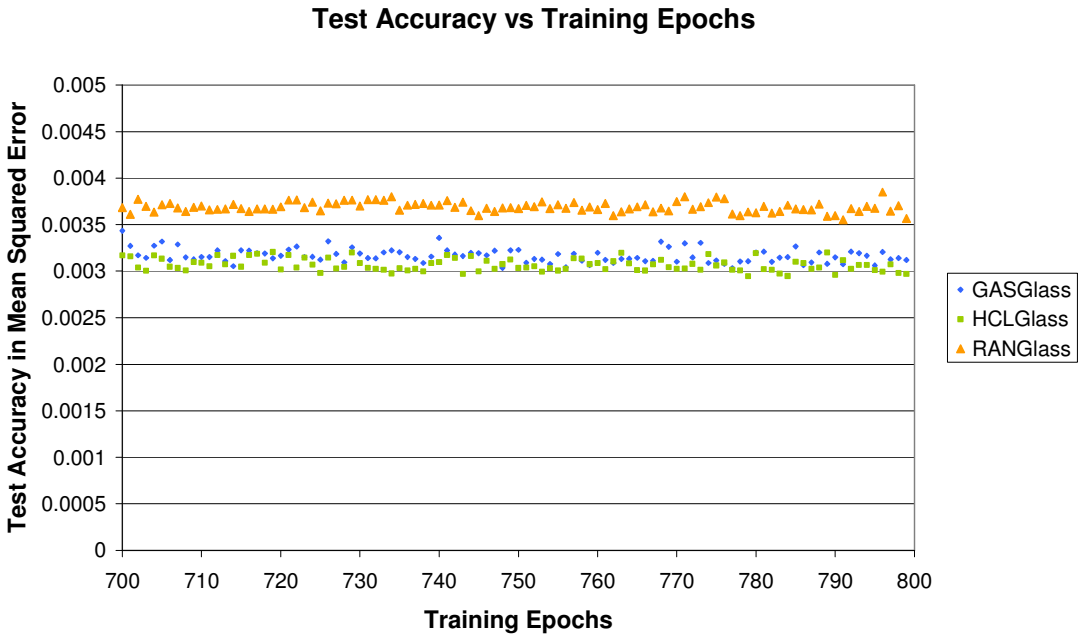


Figure 4.11: Average test accuracy for the last 100 epochs for the 800-epoch trained experiments

Overtraining

It can be observed by examining Figure 4.12 that after 250 training epochs, the Resource Allocating Network comes to an adequate understanding of the Mackey-Glass dataset. This observation can be made, as the squared error evens out to around 3.57×10^{-3}. The associated confidence factor of this trend indicates that this measurement is stable.

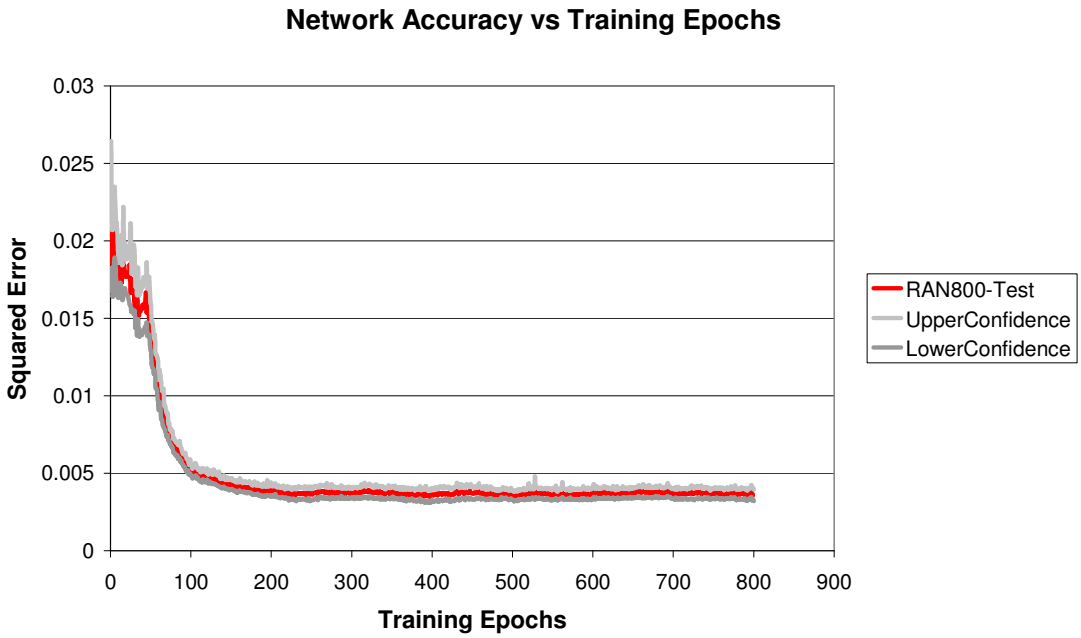


Figure 4.12: Average squared error of RAN on test set, trained over 800 epochs

4.2.2 Training speed with respect to time

Figure 4.13 and Figure 4.14 shows the amount of time in milliseconds how long on average it took to train the networks using the Mackey-Glass and the Flare datasets. Figure A. 6 provides proof of performance on the Building dataset, and is provided in Appendix A. Figure 4.15, Figure 4.16, and Figure A. 5 have also been included to clarify the aforementioned figures.

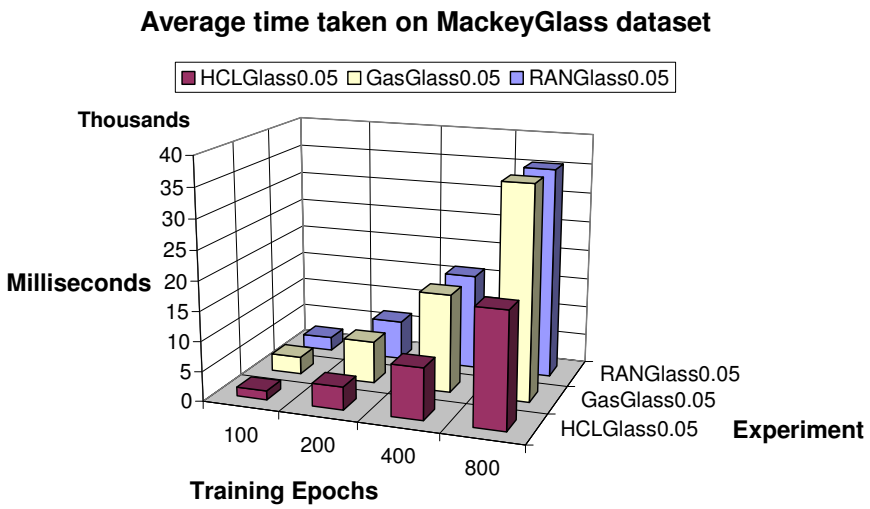
Comparison by Training Methods Used

Figure 4.13 indicates a distinct similarity in training time between the default and neural gas training strategies, when trained on the Mackey-Glass dataset. When these Mackey-Glass training times are compared to the Flare dataset (Figure 4.14) it appears that this pattern is less prominent. A noticeable difference can be seen in the rate of increase in the training times in the same figure, and accentuated on examination of the Building dataset training times (Figure A. 4).

In all the experiments, the time taken to train HCL-trained networks is substantially faster than the neural gas or default trained strategies. This could be related to the design aspects of the algorithms, where HCL needs to loop $((1/2)n)+i$ times, whereas the RAN needs to loop $n*i$ times, where n is the amount of hidden units installed, and i is the number of inputs into the network.

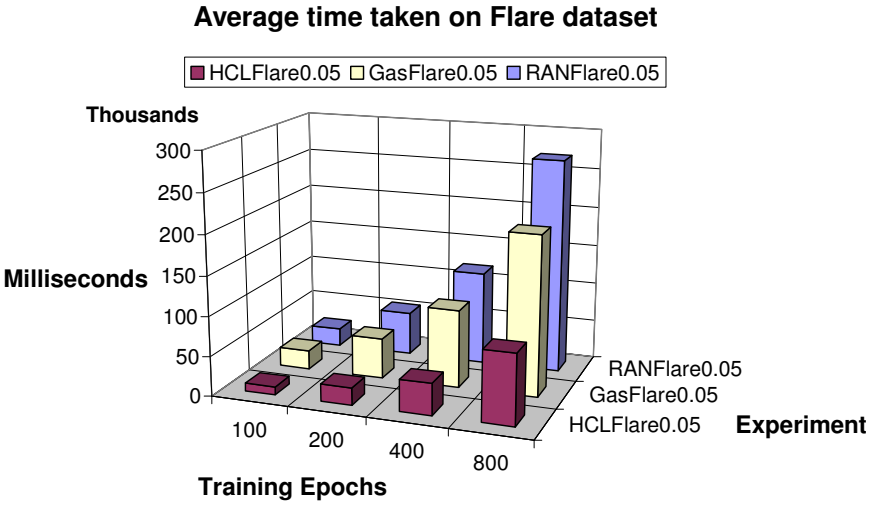
Comparison by Dataset Used

When examining the training time results, it can also be seen that the time it takes to train the network differs depending on which dataset is in use. This is expected, as each dataset is of different size, and the change in training time could easily be considered proportional to the number of input patterns in each dataset.



	100	200	400	800
HCLGlass0.05	1.49E+03	3.85E+03	8.61E+03	1.93E+04
GasGlass0.05	2.92E+03	7.09E+03	1.65E+04	3.57E+04
RANGlass0.05	2.34E+03	6.70E+03	1.62E+04	3.54E+04

Figure 4.13: Average time taken on the Mackey-Glass dataset



	100	200	400	800
HCLFlare0.05	1.00E+04	2.16E+04	4.00E+04	8.83E+04
GasFlare0.05	2.40E+04	5.08E+04	9.83E+04	2.01E+05
RANFlare0.05	2.26E+04	5.52E+04	1.18E+05	2.70E+05

Figure 4.14: Average time taken on the Flare dataset

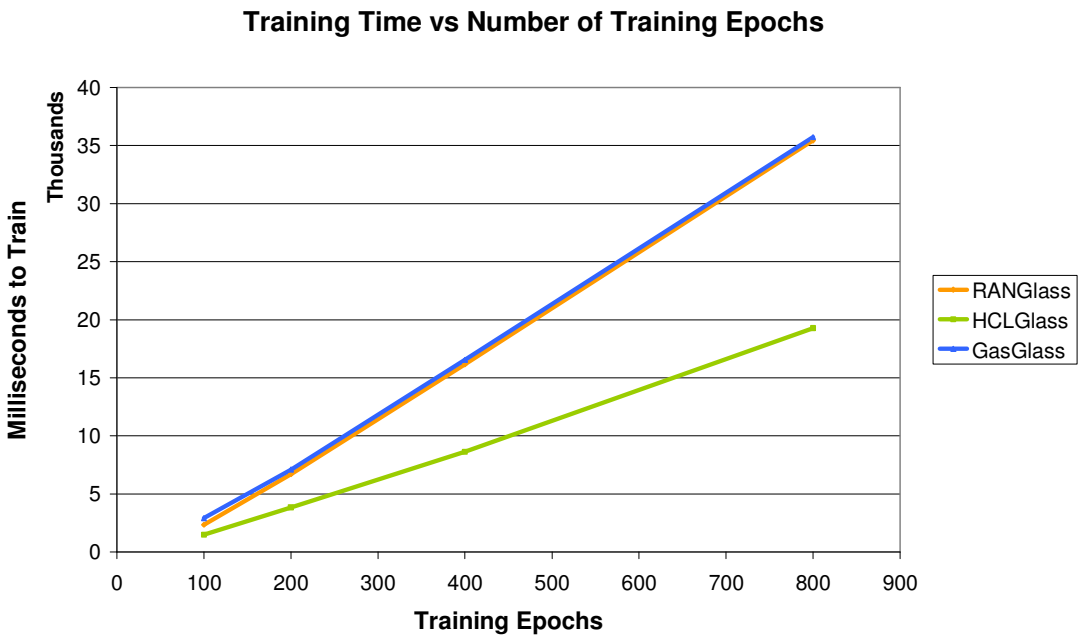


Figure 4.15: Time required to train on Mackey-Glass dataset

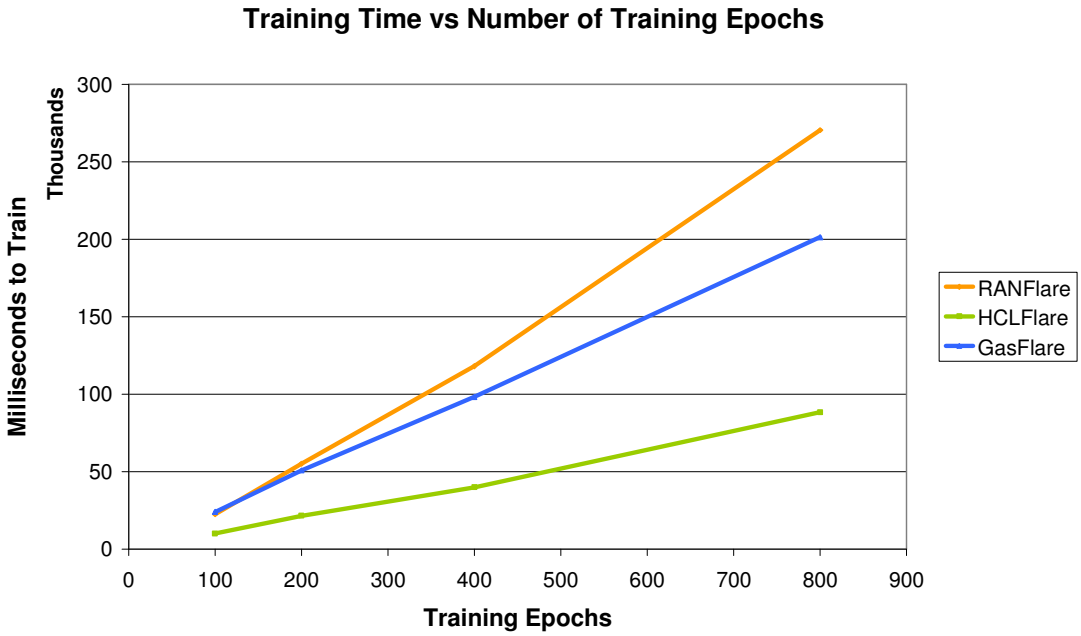


Figure 4.16: Time required to train on the Flare dataset

4.3 Combined Findings

Figure 4.17 contrasts two of the three training methods with respect to resultant network size and network error after 800 training epochs. The contrast between both the default-trained network size and the HCL trained network size is clear, where the default trained networks error fails to respond to the extra hidden units being added.

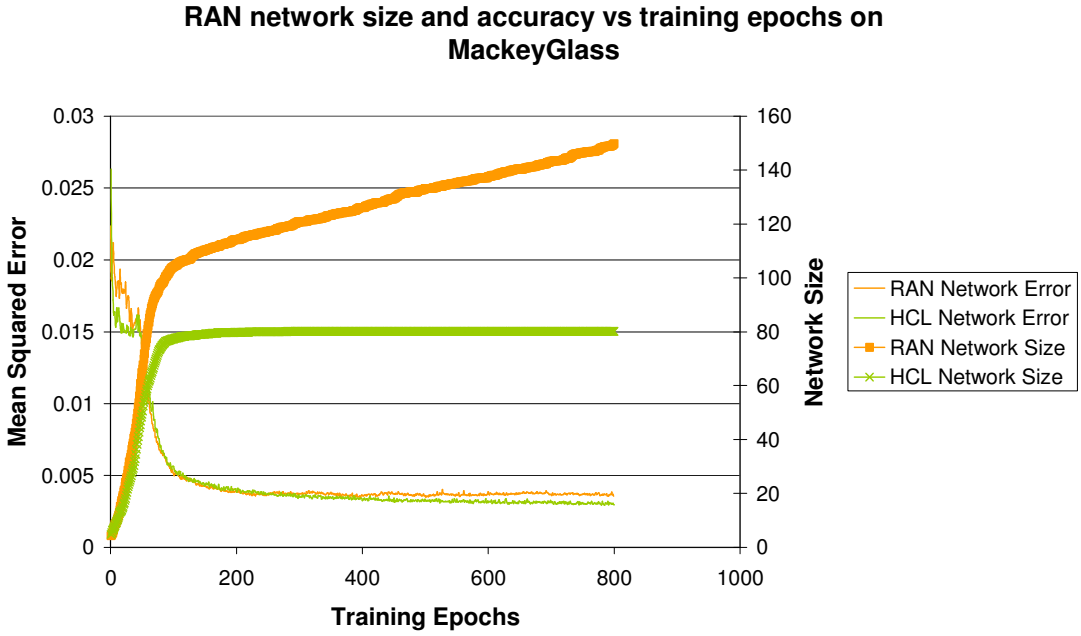


Figure 4.17: RAN and HCL Network metrics combined

5 Conclusions and Further Work

5.1 Conclusions

5.1.1 Network Size

From the figures presented, the resultant network size for the non-competitively trained RAN was indeterminate; although there was a point at which a noticeable reduction of units occurred in each of the experiments. From the point at which the growth rate was reduced onwards, the network grew only a unit every few epochs. For the competitively trained networks, each experiment showed that the solution had been acceptably learnt, by reaching a self enforced plateau. In these terms it can be said that competitive training is preferable with respect to network size over the default training method of the Resource Allocating Network. Of the competitive learning strategies tested, Hard Competitive Learning resulted in fewer units installed than by that of the Neural Gas training technique.

5.1.2 Network Accuracy

There was no dominant training technique with respect to the overall network accuracy, which was an unexpected result.

5.1.3 Processing Time Required

Though the method used was inherently inaccurate, the average trend of time required to complete an experiment was clear. Evidence collected indicates that the time required to complete an iteration increases almost linearly with the number of times it had been trained. This was explained by indicating that there was already a trend between network size and the number of training epochs the network had been trained over. The default RAN trained algorithm took the most time to complete an iteration, because it had a considerably bigger

network size. Therefore, it can be concluded that that because the processing time required to complete an iteration being inherently linked to the size of the network, it still takes longer to process larger sized networks. The fastest networks to train were the Hard Competitive Learning, and the Neural Gas trained networks had either comparable or better performance to the default trained RAN.

5.2 Further Work

As the findings related to network accuracy are inconclusive, some further work must be done to clarify the results and draw reasons as to why the network accuracy did not seem to vary to the degree that the network size did. Other work may also be done in order to improve the quality of the remaining results.

Train on more datasets

By limiting the datasets used more time was available for other experimentation. Through these experiments it has been determined that the application of competitive learning strategies to a single time series dataset, and two function approximation datasets have produced favorable results. Converting the code in order to be able to handle binary classification tasks could also yield favorable results, as casual early testing initially indicated that binary classification tasks were possible using the Iris dataset (Newman et al. 1998).

Datasets with other characteristics could also be used to train the methods, specifically those with varying amounts of controlled noise. The application of noise to the training data may be able to force a distinction in the resultant network accuracies, such that one method of training could be called superior. Also for application to real world tasks, one should not discount the effect that noisy data can have on a system.

Implement more measures of error

Investigation could be undertaken to examine the differences of results in terms of other measures of error. This may be able to separate the network accuracies of the non-competitively and competitively trained networks that have been relatively equal in this study. One variation to apply could be the normalized means squared error.

Implementation of other Competitive Learning techniques

The application of other competitive learning strategies to the Resource Allocating Network could be done in order to more comprehensively test the hypotheses of this work. Since hard competitive learning was successful, the application rival penalized learning to the RAN could yield positive results.

Application of Competitive Learning to other RANs

To demonstrate that the success had in some areas was not due to the parameters supplied to the default RAN, work could be done on varying the default parameters. The application of competitive learning to one of the extended-RANs (Rosipal et al. 1998; Wallace et al. 2004) could also demonstrate this point.

Application of different learning rates to RAN

Experiments were conducted with the RAN output layer learning rates set to either 0.05 or 0.02. Unfortunately there was simply not enough time left to collate and analyze all the data, so that only the experiments having the learning rate of 0.05 have been published. It was hoped that running these extra experiments could demonstrate that the training of the output layer was not the only variable in the calculation of network accuracy. In this vein, more analysis could be done to this already prepared data and should any anomalies or

inconsistencies be found investigation could be made at to what effect the output layer learning rate would have on the overall system.

Summary of Further Work

Both the fields of competitive learning and Resource Allocating Networks are relatively well established, yet there has been no extensive work done linking the two. As such, there are several variables that could have a potential bearing on the results presented in this project. The further work is intended to begin thought to which of the many variables could have the biggest impact on the network size and test set accuracy of competitively trained RANs.

6 References

ASHRAE 1993, *"The Great Energy Predictor Shootout" - The First Building Data Analysis And Prediction Competition*, Co-chaired by Jan F. Kreider and Jeff S. Haberl, Denver, Colorado.

Bao, HT 2000, *Knowledge Discovery and Data Mining Techniques and Practice*, http://www.netnam.vn, viewed 21 September 2005.

Bellman, R 1961, *Adaptive Control Processes: A Guided Tour*, Princeton University Press.

Bogacz, R and Giraud-Carrier, C 1995, 'Supervised Competitive Learning for Finding Positions of Radial Basis Functions', paper presented to 3rd Conference on Neural Networks and Their Applications, Kule.

Campbell, C 1997, 'Constructive Learning Techniques for Designing Neural Network Systems', University of Bristol.

Cover, TM 1965, 'Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition', *IEEE Transactions on Electronic Computers*, vol. 14, no. 3, pp. 326-34.

Cun, YL, Denker, JS and Solla, SA 1990 'Optimal brain damage ', in *Advances in neural information processing systems 2* Morgan Kaufmann Publishers Inc., pp. 598-605

de Castro, LN and Zueban, FJV 2001, 'An immunological Approach to Initialize Centers of Radial Basis Function Neural Networks', paper presented to Proceedings of V Brazilian Conference on Neural Networks, Brazil, 2-5 April 2001.

Fahlman, S and Lebiere, C 1990, 'The Cascade-Correlation Learning Architecture', in DS Touretzky (ed.), *Advances in Neural Information Processing Systems 2*, Morgan Kaufmann Publishers, Los Altos, CA, pp. 524-32.

Fritzke, B 1992, 'Growing Cell Structures - a Self-Organizing Network in k Dimensions', paper presented to Artificial Neural Networks, 2, Amsterdam, Netherlands.

Fritzke, B 1994, 'Fast Learning with Incremental RBF Networks', *Neural Processing Letters*, vol. 1, no. 1.

Fritzke, B 1995, 'A growing neural gas network learns topologies', in G Tesauro, DS Touretzky & TK Leen (eds), MIT Press, pp. 625--32.

Fritzke, B 1997, *Some Competitive Learning Methods Draft Version*, Institute for Neural Computation, Ruhn-Universitat Bochum.

Green, PJ and Silverman, BW 1994, *Nonparametric Regression and Generalized Linear Models: A roughness penalty approach*, London: Chapman & Hall.

Hassibi, B and Stork, DG 1993, *Second Order Derivatives for Network Pruning: Optimal Brain Surgeon*, vol. 5, Advances in Neural Information Processing Systems, Morgan Kaufmann, San Mateo, CA.

Jang, R 1992, *Mackey-Glass time series using 4th-order Runge-Kutta method*, viewed 7 September 2005, <http://neural.cs.nthu.edu.tw/jang/dataset/mg/mg.c>.

Kwok, T-Y and Yeung, D-Y 1995, *Constructive Feedforward Neural Networks for Regression Problems: A Survey*, The Hong Kong University of Science & Technology, Kowloon.

Levin, AU, Leen, TK and Moody, JE 1994, 'Fast Pruning Using Principal Components', paper presented to Advances in Neural Information Processing Systems.

Loos, HS 1998, *DemoGNG Code*, viewed 7 October 2005, <http://www.neuroinformatik.ruhr-uni-bochum.de/ini/VDM/research/gsn/DemoGNG/DemoGNGcode.html>.

Mackey, J and Glass, L 1977, 'Oscillation and chaos in physiological control systems', *Science*, vol. 197, no. 4300, pp. 287-9.

MacQueen, JB 1967, 'Some Methods for classification and Analysis of Multivariate Observations', paper presented to 5th Berkeley Symposium on Mathematical Statistics and Probability, Berkeley.

Martinetz, TM and Schulten, KJ 1991, 'A "neural-gas" network learns topologies', in KM T. Kohonen, O. Simula, and J. Kangas (ed.), *Artificial Neural Networks*, Elsevier, North-Holland, Amsterdam, pp. 397-402.

McCulloch, W and Pitts, W 1943, 'A logical calculus of the ideas immanent in nervous activity', *The bulletin of Mathematical Biophysics*, vol. 5, pp. 115-33.

Minsky, M and Papert, S 1969, *Perceptrons*, MIT Press, Cambridge MA.

Moody, J and Darken, C 1989, 'Fast Learning in Networks of Locally-Tuned Processing Units', *Neural Computation*, vol. 1, no. 2, pp. 281-94.

Newman, DJ, Hettich, S, Blake, CL and Merz, CJ 1998, *UCI Repository of machine learning databases*, University of California, Department of Information and Computer Science, <http://www.ics.uci.edu/~mlearn/MLRepository.html>.

Parekh, R, Yang, J and Honavar, V 2000, 'Constructive Neural-Network Learning Algorithms for Pattern Classification', *IEEE Transactions on Neural Networks*, vol. 11, no. 2, pp. 436 - 50.

Platt, J 1991, 'A Resource-Allocating Network for Function Interpolation', *Neural Computation*, vol. 3, no. 2, pp. 213-25.

Prechelt, L 1994, *PROBEN1 - A Set of Benchmarks and Benchmarking Rules for Neural Network Training Algorithms*, University Karlsruhe.

Prechelt, L 1996, 'Investigation of the CasCor Family of Learning Algorithms', *Neural Networks*, vol. 10, no. 5, pp. 885-96.

Rosenblatt, F 1958, 'The Perceptron: A Probabilistic Model for Information Storage And Organization in the Brain', *Psychological Review,*, vol. 65, no. 6, pp. 386-408.

Rosipal, R, Koska, M and Farkas, I 1998, 'Prediction of Chaotic Time-Series with a Resource-Allocating RBF Network', *Neural Processing Letters*, vol. 7, pp. 185-97.

Rumelhart, D and Zipser, D 1985, 'Feature Discovery By Competitive Learning', *Cognitive Science*, vol. 9, pp. 75-112.

Wallace, M, Tsapatsoulis, N and Kollias, S 2004, 'Intelligent initialization of resource allocating RBF networks', *Neural Networks*, vol. 18, pp. 117-22.

Weigend, AS, Rumelhart, D. E., & Huberman, B. A. 1991, 'Generalization by weight-elimination with application to forecasting', paper presented to Advances in Neural Information Processing Systems, San Mateo, CA.

Werbos, P 1974, 'Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.' PhD thesis.

Widrow, B and Hoff, M 1960, 'Adaptive Switching Circuits', paper presented to 1960 IRC WESCON Convention Record, New York.

Appendix A: Additional Supporting Graphs

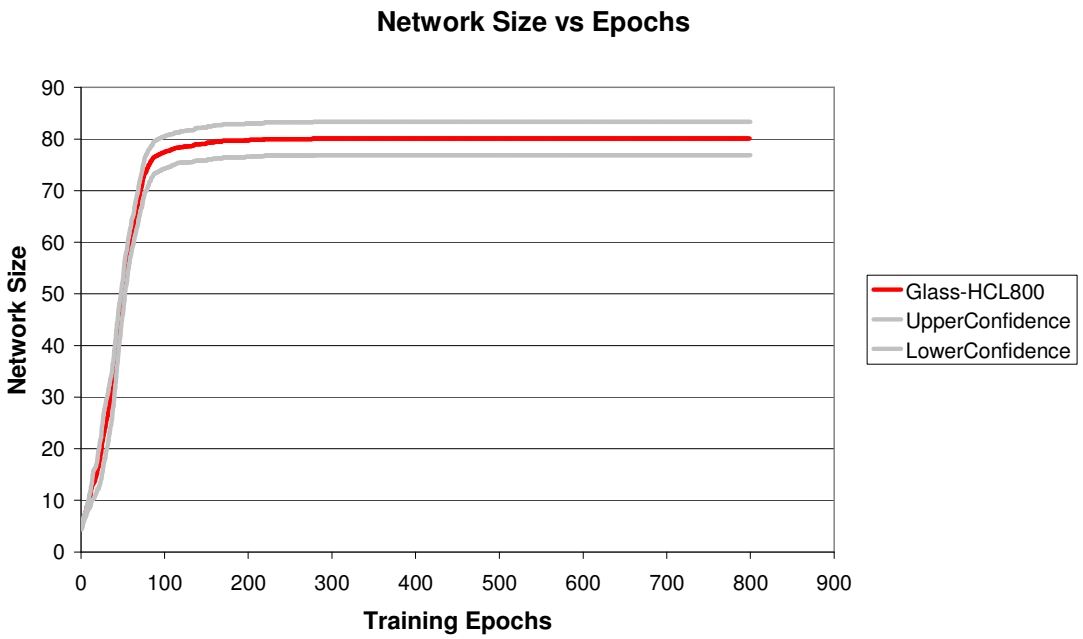


Figure A. 1: Hard competitive learning network size versus training epochs

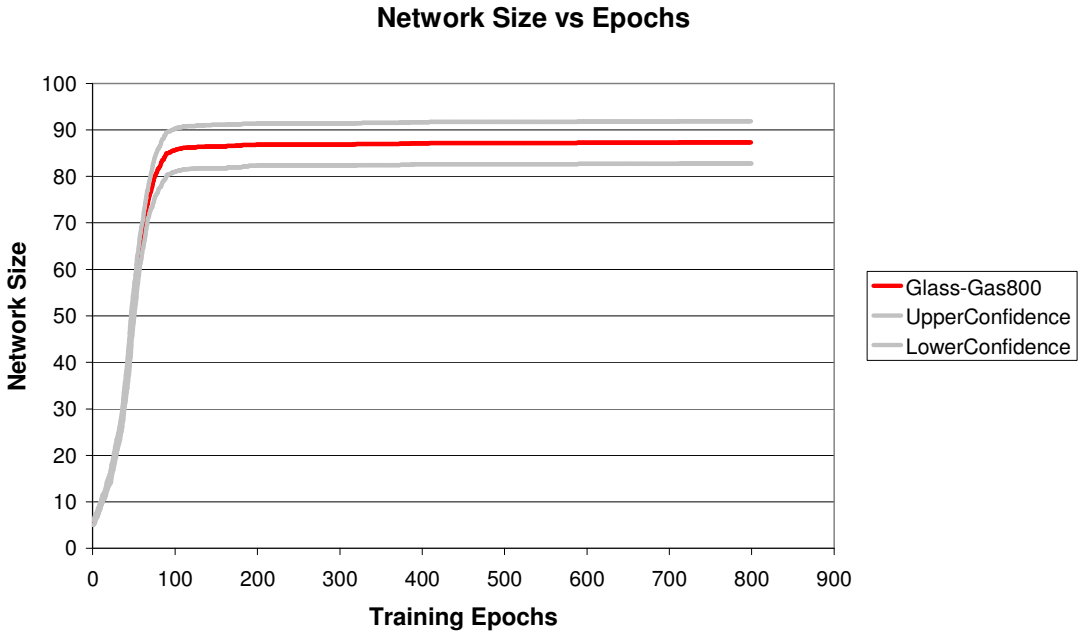


Figure A. 2: Neural gas network size versus training epochs

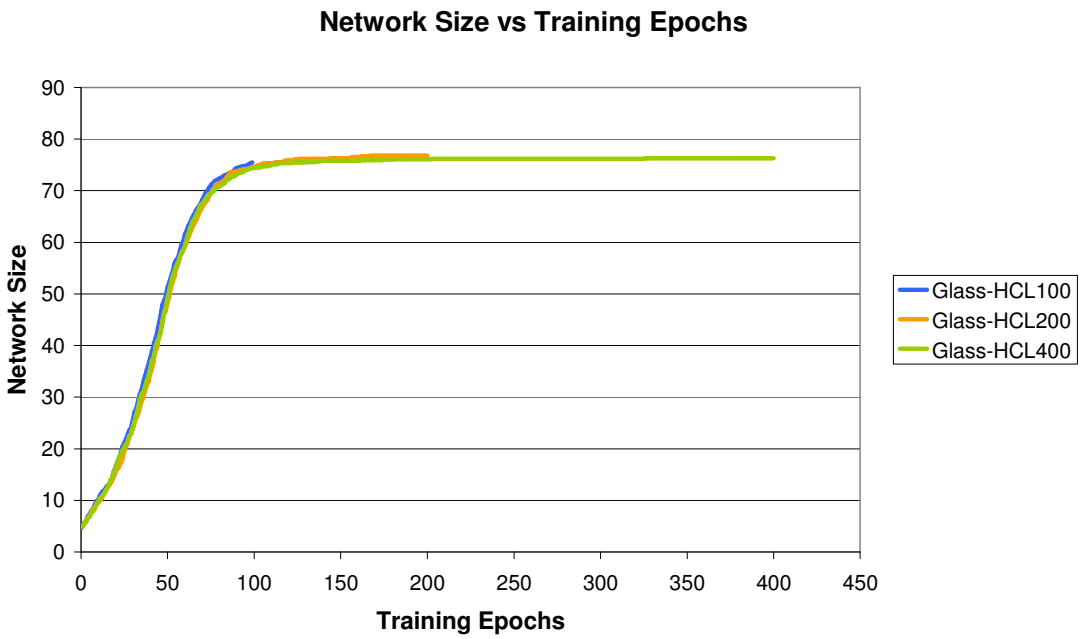
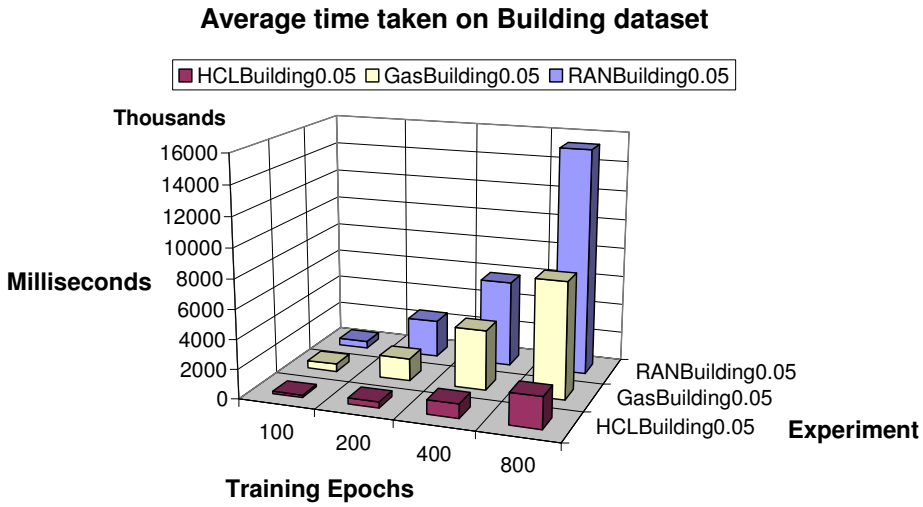
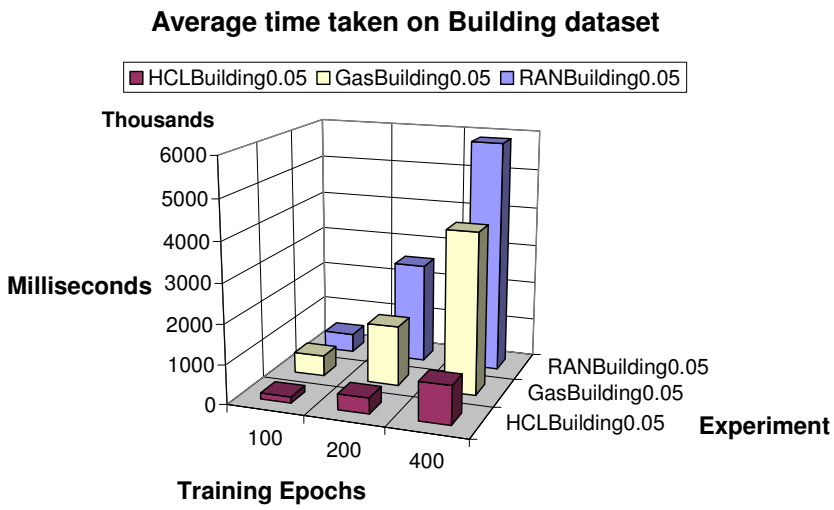


Figure A. 3: Comparison of hard competitive learning trained networks over 100, 200 & 400 training epochs



	100	200	400	800
HCLBuilding0.05	1.61E+05	4.02E+05	9.71E+05	2.13E+06
GasBuilding0.05	5.26E+05	1.52E+06	4.08E+06	7.89E+06
RANBuilding0.05	4.80E+05	2.56E+06	5.86E+06	1.53E+07

Figure A. 4: Average time taken on the Building dataset



	100	200	400
HCLBuilding0.05	1.61E+05	4.02E+05	9.71E+05
GasBuilding0.05	5.26E+05	1.52E+06	4.08E+06
RANBuilding0.05	4.80E+05	2.56E+06	5.86E+06

Figure A. 5: Average time taken on the Building dataset over 100, 200 and 400 training epochs

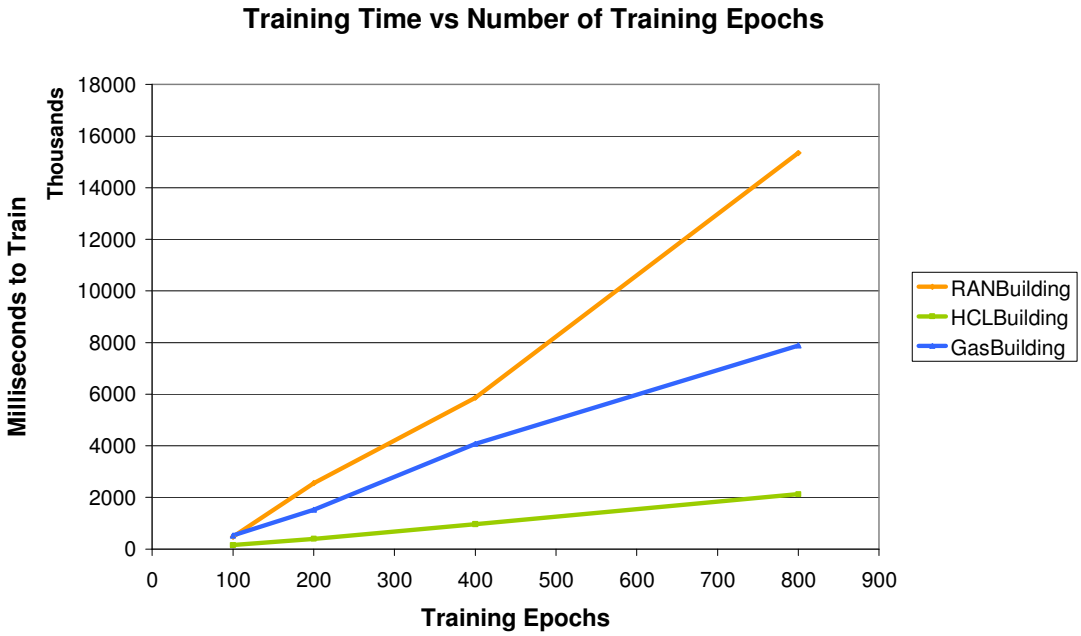


Figure A. 6: Time required to train on the Building dataset

Appendix B: Dataset Descriptions

B.1 The Flare Dataset

Index	*Label*	*Units*	*Attribute Type*	*Min – Max*	*Standard Deviation ± Error*
Input 1	Modified Zurich Class Code 1	Binary	Discrete	0 - 1	0.00E+00 ± 0.00E+00
Input 2	Modified Zurich Class Code 2	Binary	Discrete	0 - 1	3.45E-01 ± 1.06E-02
Input 3	Modified Zurich Class Code 3	Binary	Discrete	0 - 1	3.99E-01 ± 1.22E-02
Input 4	Modified Zurich Class Code 4	Binary	Discrete	0 - 1	4.17E-01 ± 1.28E-02
Input 5	Modified Zurich Class Code 5	Binary	Discrete	0 - 1	2.85E-01 ± 8.73E-03
Input 6	Modified Zurich Class Code 6	Binary	Discrete	0 - 1	1.97E-01 ± 6.03E-03
Input 7	Modified Zurich Class Code 7	Binary	Discrete	0 - 1	4.63E-01 ± 1.42E-02
Input 8	Largest spot size code 1	Binary	Discrete	0 - 1	3.43E-01 ± 1.05E-02

Input 9	Largest spot size code 2	Binary	Discrete	0 - 1	4.04E-01 ± 1.24E-02
Input 10	Largest spot size code 3	Binary	Discrete	0 - 1	4.88E-01 ± 1.49E-02
Input 11	Largest spot size code 4	Binary	Discrete	0 - 1	4.02E-01 ± 1.23E-02
Input 12	Largest spot size code 5	Binary	Discrete	0 - 1	1.57E-01 ± 4.81E-03
Input 13	Largest spot size code 6	Binary	Discrete	0 - 1	2.03E-01 ± 6.23E-03
Input 14	Spot distribution code 1	Binary	Discrete	0 - 1	4.63E-01 ± 1.42E-02
Input 15	Spot distribution code 2	Binary	Discrete	0 - 1	4.97E-01 ± 1.52E-02
Input 16	Spot distribution code 3	Binary	Discrete	0 - 1	4.07E-01 ± 1.25E-02
Input 17	Spot distribution code 4	Binary	Discrete	0 - 1	1.78E-01 ± 5.46E-03
Input 18	Activity (current status)	Binary	Discrete	0 - 1	3.61E-01 ± 1.11E-02
Input 19	Evolution (status W.R.T time)	Float	Continuous	0 - 1	3.10E-01 ± 9.50E-03

Input 20	24 hour activity code	Float	Continuous	0 - 1	1.60E-01 ± 4.91E-03
Input 21	Historically complex	Binary	Discrete	0 - 1	4.91E-01 ± 1.50E-02
Input 22	Recent historically complex change	Binary	Discrete	0 - 1	3.31E-01 ± 1.01E-02
Input 23	Area of flare	Binary	Discrete	0 – 1	1.57E-01 ± 4.81E-03
Input 24	Area of largest spot	Binary	Discrete	0 – 1	0.00E+00 ± 0.00E+00
Output 1	Common severity flares produced in last 24 hours	Float	Continuous	0 - 1	1.04E-01 ± 3.20E-03
Output 2	Moderate severity flares produced in last 24 hours	Float	Continuous	0 – 1	3.79E-02 ± 1.16E-03
Output 3	Severe severity flares produced in last 24 hours	Float	Continuous	0 - 1	4.32E-02 ± 1.32E-03

Table B. 1: Description of the Flare dataset

B.2 The Building Dataset

Index	*Label*	*Units*	*Attribute Type*	*Min – Max*	*Standard Deviation ± Error*
Input 1	Month	Integer	Discrete	1 – 12	4.30E+00 ± 6.62E-02
Input 2	Day	Integer	Discrete	1 – 31	8.68E+00 ± 1.34E-01
Input 3	Year	Integer	Discrete	89 – 90	4.60E-01 ± 7.10E-03
Input 4	Hour	Integer	Discrete	0 – 2300	6.92E+02 ± 1.07E+01
Input 5	Temperature	Degrees Fahrenheit	Continuous	2.6 – 99.5	1.62E+01 ± 2.50E-01
Input 6	Humidity	Percent	Continuous	0 – 0.0222	5.00E-03 ± 7.71E-05
Input 7	Solar Flux	Watts / meter^2	Continuous	-0.7 – 1025	2.39E+02 ± 3.68E+00
Input 8	Wind speed	Miles / hour	Continuous	0 -26.28	3.51E+00 ± 5.42E-02
Output 1	Total Energy Used	Kilowatts	Continuous	374.32 – 995.34	1.52E+02 ± 2.34E+00
Output 2	Cold Water Consumption	Millions Btu / hour	Continuous	0 – 8	1.14E+00 ± 1.76E-02
Output 3	Hot Water Consumption	Millions Btu / hour	Continuous	0 – 6.3	1.42E+00 ± 2.18E-02

Table B. 2: Building dataset description

Appendix C: Program Arguments

Parameter	*Values*	*Default*	*Description*
Dataset	Mackey-Glass, Flare.txt, Building1.txt	N/A	Describes which dataset to use in training and testing.
Center Movement Strategy	RAN, GAS, HCL	N/A	Specifies which training method will be used to train the network. RAN is the default strategy, GAS is Neural Gas and HCL is Hard Competitive Learning
Name	Any string, not allowed spaces, or characters that are not permitted by the local computers file system.	N/A	Defines a name for the experiment. This is used directly in all files produced by the experiment.
iterations	Any integer >0	N/A	Controls how many times the experiment is actually carried out (used for statistical validation)

err	0.05, 0.02	0.05	Err represents the learning rate to use- how much freedom the output layer weights have in training
epochs	Any integer >0	400	Specifies how many presentations of the training set are to be made
draw	No arguments required	Do not draw	The first network trained will have a graph outputted for each training epoch. This parameter does not have any arguments to it.

Table C. 1: Parameters used when specifying an experiment